68' MICRO JOURNAL

Australia	A $ 4.75	New Zealand	NZ $ 6.50
Singapore	S $ 9.45	Hong Kong	H $23.50
Malaysia	M $ 9.45	Sweden	30:-SEK

$2.95USA

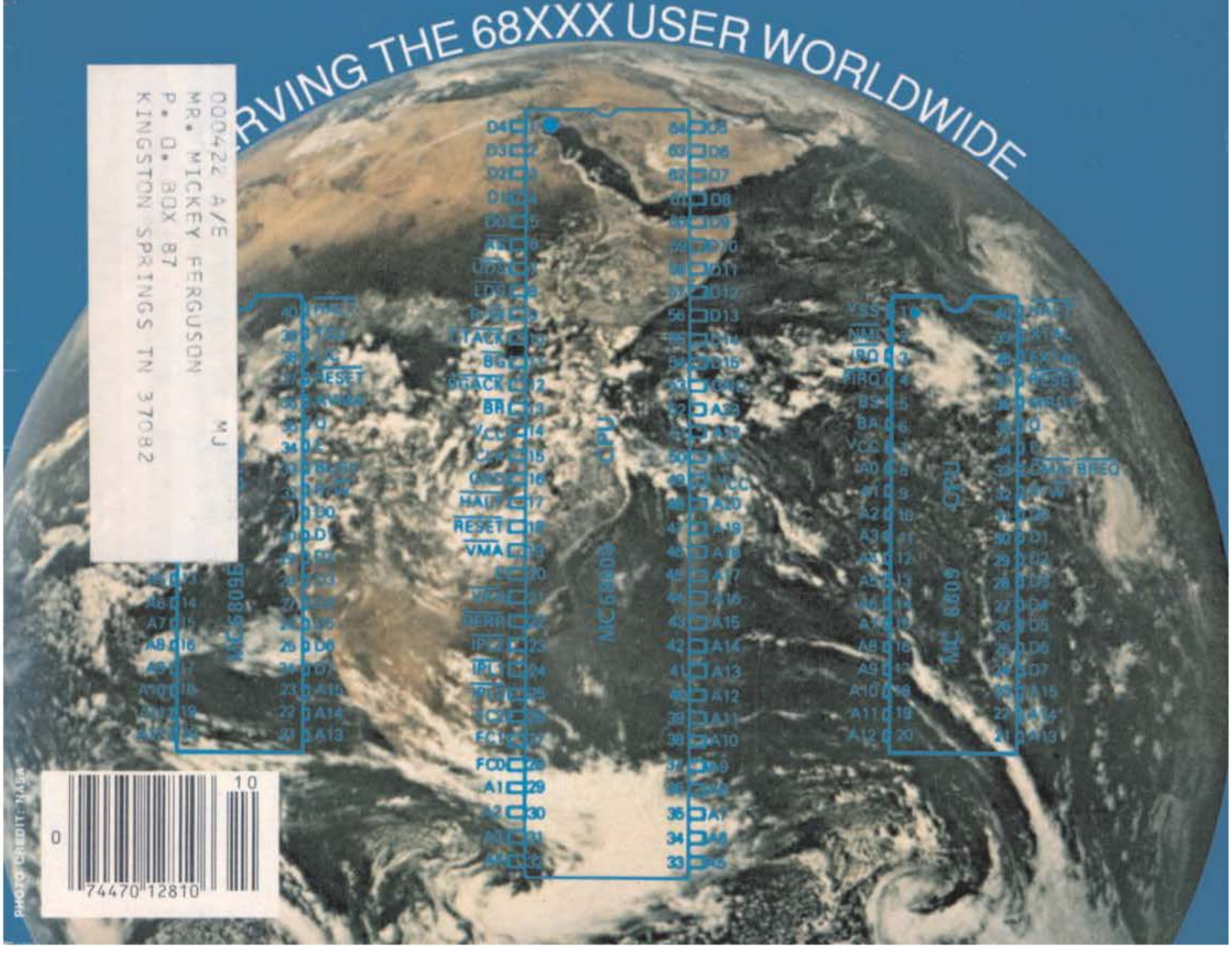
Motorola VME-MACINTOSH-S 50 & Other 68XXX Systems
6809 68008 68000 68010 68020 68030
OS-9 The Magazine for Motorola CPU Devices FLEX SK*DOS
A User Contributor Journal

This Issue:

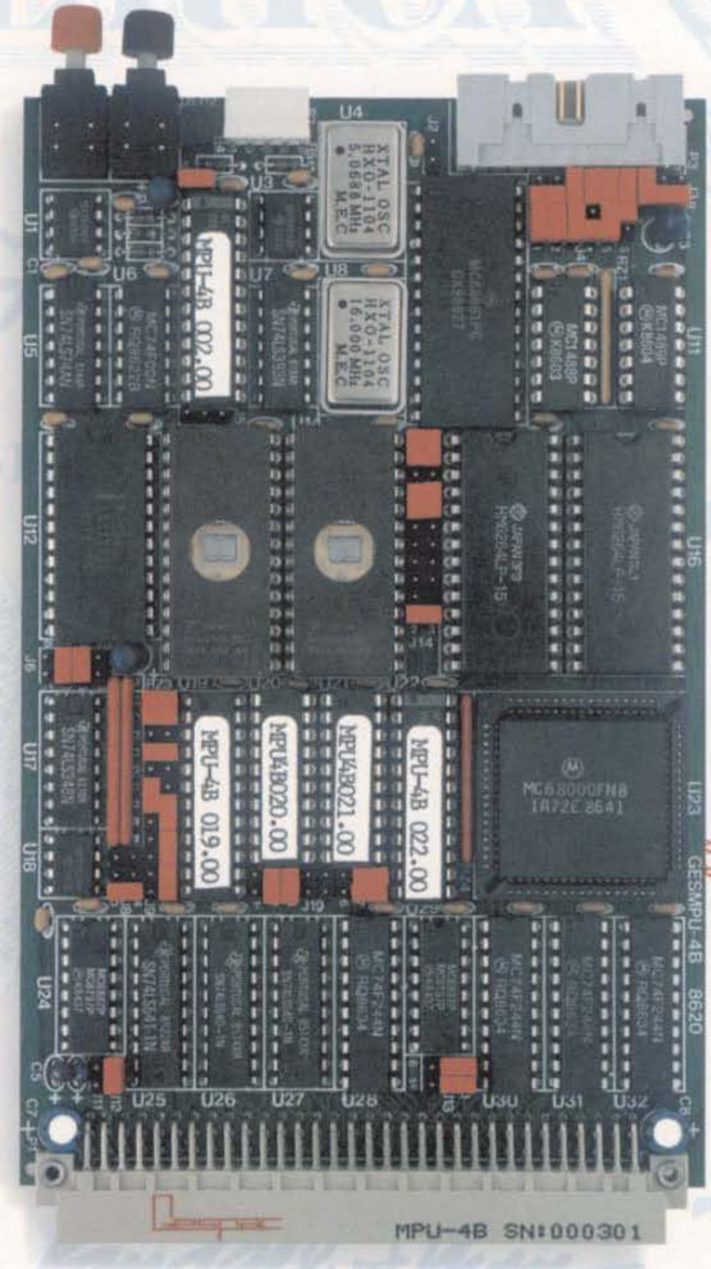
Plus: Password - Text Hacking & Logically Speaking And Lots More!

VOLUME IX ISSUE X ● Devoted to the 68XXX User ● October 1987

The Grandfather of "DeskTop Publishing™"



GESPAC Gives You 68000 Performance at 8-bit Prices.



Actual Size

Introducing another Price / Performance breakthrough from GESPAC: A complete 68000 CPU module for $395 unit price. The GESMPU-48.

Wherever you thought of using an 8-bit microprocessor to do a job, you con now use the 16/32-bit GESMPU-48 instead. It will do the job, better, faster, and best-of-all for the same money.

On a single height Eurocard, we have packed an 8 MHz 68000 microprocessor (16 MHz optional), four sockets for up to 64 Kilobytes of zero-wait-states CMOS RAM and up to 128 Kilobytes of EPROM, one RS-232 serial port, and three 16-bit timers.

The GESMPU-48 is fully expandable through the standard G-64 bus, to accommodate up to 16 Megabytes of external memory. You can add any of more than 300 I/O modules available from GESPAC and a growing numbers of independent G-64 bus vendors.

To make your programming tasks easier, GESPAC supports the GESMPU-48 with the OS-9® and PDOS® real-time, multi-tasking operating systems and most popular high level languages and software development tools.

$395.00
Single Quantity

$316 @ 100 pieces

If you too like the idea of getting more for less, contact us today to receive information on the GESMPU-48 and the G-64 bus concept from GESPAC—the leader in single Eurocard microcomputer products worldwide.

Call Toll Free 1-800-4-GESPAC or Call (602) 962-5559.

IN USA - CANADA
50A West Hoover Ave.
Mesa, Arizona 85202
Tel. (602) 962-5559
Telex 386575

INTERNATIONAL
3, chemin des Aulx
CH-1228 Geneva
Tel. (022) 713400
Telex 429989

G-64 and G-96 are registered trademarks of GESPAC SA OS-9 is a registered trademark of Microware Corp. PDOS is a registered trademark of Eyring Research Institute Inc

THE GMX 020BUG DEBUGGER/DIAGNOSTIC PACKAGE

This extensive firmware package provides a broad range of program development tools and a complete suite of diagnostic programs for exercising GMX Micro-20 hardware.

The debugger includes commands for displaying and modifying registers and memory. If the optional 68881 Floating-Point Coprocessor is installed, its registers are also accessible. Memory can be displayed in hexadecimal and ASCII format, as floating-point values (single, double, extended or packed format), or as disassembled instructions (including FPC instructions). Memory modify can be done with hexadecimal values, with ASCII strings, with floating-point values, or with a one-line assembler which supports the full 68020 instruction set (although not the FPC instructions). Block move, fill, and search are also available.

Several different modes for tracing or executing user programs are provided, along with a powerful breakpoint facility. Programs and data may be downloaded from a host system or uploaded back to the host, and the GMX Micro-20 console may be used as a host system terminal. A serial printer may be hooked up, and used to make hardcopy listings of debugger sessions as desired.

The diagnostic firmware includes 90 test commands and 16 utilities. Complete test suites are provided for each functional block of the GMX Micro-20's hardware, including, for example, 9 different tests for memory, 9 tests for serial I/O ports, 2 tests for the 68881 FPC, and 9 tests for the optional memory management unit. For the peripheral control interfaces (floppy disk, SASI/SCSI hard disk or tape), test commands support a broad range of peripheral operations (read, write, format, etc.) so that the user may test both the interface and an attached device. Tests are provided for add-on I/O boards, including the ARCnet interface, I/O Channel interface, and parallel and serial expansion boards.

The utility commands allow the user to execute groups of test commands conveniently, repeat commands or command groups, enable or disable detailed fault reporting, count detected errors, or execute all the non-peripheral tests as a group. A switch option allows this last function to be invoked automatically at power-on or reset. Other utilities allow the user to check the state of the various jumpers and switches on the GMX Micro-20 directly.

In addition to the Diagnostic command package, 020Bug contains a confidence test which is always run after power-on or reset. This test does a quick checkout of the processor and the basic system elements that are needed for 020Bug operation. If any defect is found, an error code is signalled by on-and-off blinks of an LED.

INCLUDED AT NO EXTRA COST WITH THE GMX MICRO-20

DEBUGGING COMMANDS

MD	— Memory display
MM	— Memory modify
MS	— Memory set
BF	— Block fill
BM	— Block move
BS	— Block search
RD	— Register display
RM	— Register modify
OF	— Offset registers
BR	— Breakpoint set
NOBR	— Breakpoint delete
G	— Go to target code
GD	— Go, delete breakpoints
GN	— Go, stop after 1 instruction
GT	— Go, set temp breakpoint
T	— Trace
TC	— Trace on change of flow
TT	— Trace to temp breakpoint
LO	— Download
DU	— Upload
VE	— Verify download
TM	— Terminal mode
PA	— Printer attach
NOPA	— Remove printer
PF	— Port format
TD	— Time display
TS	— Time set
SD	— Switch directory
RS	— Restart system
OS	— Boot operating system

UTILITY COMMANDS

NV	— Non-verbose mode
SE	— Stop on error mode
LE	— Loop on error mode
LC	— Loop continual mode
ST	— Selftest mode
STL	— Selftest with LED mode
DE	— Display errors
ZE	— Zero errors
DP	— Display pass count
ZP	— Zero pass count
RL	— Read loop
WL	— Write loop
DJ	— Display baud rate jumper settings
DS	— Display switch
MJ	— Display MMU board jumper settings
SX	— Scan I/O expansion space

TEST COMMANDS

AN — ARCnet interface tests

A	— Wakeup test
B	— DIP Switch test
C	— Interrupts test
D	— Buffer test

CA20 - On chip cache tests

A	— Basic caching
B	— Unlike function codes
C	— Disable
D	— Clear

FD — Floppy disk tests

A	— Set parameters
B	— Drive select toggle
C	— Side select toggle
D	— Restore
E	— Seek
F	— Format track
G	— Read
H	— Write
I	— Copy buffer
J	— Compare buffer
K	— Fill buffer

IC — I/O Channel tests

A	— Print test pattern
B	— Bit rotate

MH — Miscellaneous hardware tests

A	— 68881 FPC instructions
B	— 68881 FPC control functions
C	— Tick generator
D	— Interrupt sources

MT — Memory tests

A	— Set function code
B	— Set start address
C	— Set end address
D	— Random inversion test
E	— March address test
F	— Walk-a-bit test
G	— Refresh test
H	— Random byte test
I	— Program test
J	— TAS test
K	— Test 0000-1FFF
L	— Partial longword writes test

MU — Memory Management tests

A	— Map RAM data test
B	— Map RAM address test
C	— Map RAM partial write test
D	— Map RAM random data test
E	— Accessed bit reset test
F	— Address mapping test
G	— Accessed/Dirty bits test
H	— Valid/Write Enable test
I	— Task size test

PP — Parallel port tests

A	— Print test pattern
B	— Continual test bit pattern
C	— Test bit pattern for 10 sec

PX — Parallel I/O expansion board tests

A	— Data, handshake, and IRQ test
B	— P4 connector test
C	— Data and handshake toggle

SA — SASI/SCSI port with SASI device

A	— Select drive
B	— Scan data lines
C	— Restore
D	— Seek
E	— Read
F	— Write
G	— Compare buffers
H	— Fill write buffer
I	— Test interrupt
J	— Park head
K	— Format

SC — SASI/SCSI port with SCSI device

A	— Select drive
B	— Scan data lines
C	— Restore
D	— Seek
E	— Read
F	— Write
G	— Compare buffers
H	— Fill write buffer
I	— Test interrupt
J	— Stop drive
K	— Format

SI — Serial I/O tests

A	— Select DUARTs
B	— Internal loopback
C	— External loopback
D	— Baud rates
E	— Parity modes
F	— Character lengths
G	— Handshake lines
I	— BREAK detect
J	— Interrupt output
K	— Continual handshake toggle

TA — Tape drive tests

A	— Rewind
B	— Read
C	— Write
E	— Compare buffers
F	— Fill write buffer
G	— Erase

GMX™ 1337 W. 37th Place, Chicago, IL 60609
(312) 927-5510 — TWX 910-221-4055 — FAX (312) 927-7352

A Member of the CPI Family

68 Micro Journal

10 Years of Dedication to Motorola CPU Users

6800 6809 68000 68010 68020

The Originator of "DeskTop Publishing™"

Publisher
Don Williams Sr.

Executive Editor
Larry Williams

Production Manager
Tom Williams

Office Manager
Joyce Williams

Subscriptions
Kristi Hart

Contributing & Associate Editors

Ron Anderson
Ron Voigts
Doug Lurie
Ed Law

Dr. E.M. "Bud" Pass
Art Weller
Dr. Theo Elbert
& Hundreds More of Us

Contents

68 MICRO JOURNAL

"Contribute Nothing - Expect Nothing" DMW 1986

COMPUTER PUBLISHING, INC.

"Over a Decade of Service"

"World Wide"

68 MICRO JOURNAL
Computer Publishing Center
5900 Cassandra Smith Road
PO Box 849
Hixson, TN 37343

Phone (615) 842-4600 Telex 510 600-6630

68 Micro Journal is the *original* "DeskTop Publishing" product and has continously published since 1978 using only micro-computers and special "DeskTop" software. Using first a kit built 6800 micro-computer, a modified "ball" typewriter, and "home grown" DeskTop Publishing software. None was commercially available at that time. For over 10 years we have been doing "DeskTop Publishing"! *We originated what has become traditional "DeskTop Publishing"!* Today 68 Micro Journal is acknowledged as the "Grandfather" of "DeskTop Publishing" technology.

68 Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage paid ISSN 0194-5025 at Hixson, TN. and additional entries. Postmaster: send form 3597 to 68 Micro Journal, POB 849, Hixson, TN 37343.

Subscription Rates

1 Year $24.50 USA, Canada & Mexico $34.00 a year. Others add $12.00 a year surface, $48.00 a year Airmail, USA funds. 2 years $42.50, 3 years $64.50 plus additional postage for each additional year.

Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephone number, date and a statement that the material is original and the property of the author. Articles submitted should be on diskette, OS-9, SK*DOS, FLEX, Macintosh or MS-DOS. All printed items should be dark type and satisfactory for photo-reproduction. No blue ink! No hand written articles - please! Diagrams o.k.

Please - do not format with spaces any text inde ts, charts, etc. (source listing o.k.). We will edit in all formas i g. Text should fall flush left and use a carriage return only o indicate a paragraph end. Please write for free authors guide.

Letters & Advertising Copy

Letters to the Editor should be the original copy, signed! Letters of grip as well as praise are acceptable. *We reserve the right to reject any letter or advertising material, for any reason we deem advisable.* Advertising Rates: Commercial please contact 68 Micro Journal Advertising Department. Classified advertising must be non-commercial. Minimum of $15.50 for first 15 words. Add $.60 per word thereafter. No classifieds accepted by telephone.

EXCITING SOFTWARE FROM THE LEADER...

microware

OS-9 ELECTRONIC MAIL

Flash your message on Electronic Mail Mail is a screen- or line-oriented program that runs on your OS-9/680X0 systems or over OS-9/NET. You can use distributed mailing lists or consecutive mailing list to get your message delivered. And received mail can be sent directly to your printer for immediate printout, spooled on a multiuser system or saved to a file. Mail features on-line help and complete, easy-to-understand documentation.
Electronic Mail $150.00.

PRINT SPOOLER

Spool it and Print it! Someone beat you to the printer? Don't blow your top while you cool your heels--get the OS-9/68000 Print Spooler and relax. The full-featured Print Spooler automatically routes and monitors the status of your devices and the output files to be spooled. Now you can have a complete print spooling management system at an affordable price.
OS-9 Print Spooler $150.00

FORTRAN

Crunch It! with Our New FORTRAN 77 Compiler Now you have a powerful new tool to take full advantage of the 68000 family of microprocessors. With Microware's FORTRAN 77 Compiler you can generate code that uses system-wide modules instead of linking redundant copies of the standard library to each program. Result: less memory, less disk space, faster loading and external updating!
FORTRAN 77 Compiler $750.00

OS-9/ST

NEW for your Atari ST! Now you can have the power of OS-9 on your Atari 520 or 1040 ST. A true multi-tasking environment for professional real-time results. OS-9/ST is available in two configurations: Personal and Professional. Choose either version for true multi-user support...And all at a price that puts UNIX to shame.

Personal OS-9/ST combines the power of OS-9 with an interactive, structured Basic. **$150.00**

Professional OS-9/ST has a powerful Assembler, Linker and User Debugger and the tools to turn your Atari ST into a full C Language workstation. **$600.00**

OS-9/68020 C COMPILER

NEW "speed demon" C Compiler! Now you can get your hands on a highly optimized C Language power tool--the OS-9/68020 C Compiler. When coupled with the MC68881 math co-processor, this compiler will let you'll blast through complex math functions in the blink of an eye. All compiler/assembler/linker options are controlled by an intelligent compiler executive that spares you from memorizing compiler options and module-calling sequences. And the compiler includes library functions for memory management and system events, and much, much more! The new OS-9/68020 C Compiler is included with the Professional OS-9/68020 System Software Package.
New C Language Compiler $750.00

To order these exciting NEW products or for more information... ***CALL TODAY!***

Microware Systems Corporation
1900 N.W. 114th Street * Des Moines, Iowa 50322
Phone 515-224-1929 * Telex 910-520-2535

West Coast Office
4401 Great American Parkway * Suite 220
Santa Clara, California 95054

Microware Japan, Ltd.
41-19 Honcho 4 Chome, Funabashi City * Chiba 273,
Japan * Phone 0473(28)4493 * Telex 781-299-3122

Micromaster Scandinavian AB
S:t Persgatan 7
Box 1309
S751-43 Uppsala
Sweden
Phone: 018-138595
Telex: 76129

Dr. Rudolf Keil, GmbH
Porphystrasse 15
D-6905 Schriesheim
West Germany
Phone: (0 62 03) 67 41
Telex: 465025

Elsoft AG
Zelweg 12
CH-5405 Baden-Dattwil
Switzerland
Phone: (056) 83-3377
Telex: 828275

Vivaway Ltd.
36-38 John Street
Luton, Bedfordshire, LU1 2JE
United Kingdom
Phone: (0582) 423425
Telex: 825115

Microprocessor Consultants
92 Bynya Road
Palm Beach 2108
NSW Australia
Phone: 02-919-4917

Microdata Soft
97 bis, rue de Colombes
92400 Courbevoie
France
Phone: 1-788-80-80
Telex: 615405

Microware is on the move. We have openings for Technical and Marketing Professionals. Send your resume (in confidence) today and find out more about these exciting opportunities.

OS-9 and BASIC09 are trademarks of Microware and Motorola. UNIX is a trademark of Bell Laboratories, Inc.

☆ MUSTANG -020 Super SBC™

DATA-COMP proudly presents the first Under $5000 "SUPER MICRO".

The MUSTANG-020™

MUSTANG-020™

The MUSTANG-020 68020 SBC provides a powerful, compact, 32 bit computer system featuring the "state of the art" Motorola 68020 "super" micro-processor. It comes standard with 2 megabyte of high-speed SIP dynamic RAM, serial and parallel ports, floppy disk controller, a SASI hard disk interface for intelligent hard disk controllers and a battery backed-up time-of-day clock. Provisions are made for the super powerful Motorola MC68881 floating point math co-processor, for heavy math and number crunching applications. An optional network interface uses one serial (four (4) standard, expandable to 20) as a 125/bit per second network channel. Supports as many as 32 nodes.

The MUSTANG-020 is ideally suited to a wide variety of applications. It provides a cost effective alternative to the other MC68020 systems now available. It is an excellent introductory tool to the world of hi-power, hi-speed new generation "super micros". In practical applications it has numerous applications, ranging from scientific to education. It is already being used by government agencies, labs, universities, business and practically every other critical applications center, worldwide, where true multi-user, multi-tasking needs exist. The MUSTANG-020 is UNIX C level V compatible. Where low cost and power is a must, the MUSTANG-020 is the answer, as many have discovered. Proving that price is not the standard for quality!

As a software development station, a general purpose scientific or small to medium business computer, or a super efficient real-time controller in process control, the MUSTANG-020 is the cost effective choice. With the optional MC68881 floating point math co-processor installed, it has the capability of systems costing many times over it's total acquisition cost.

With the DATA-COMP "total package", consisting of a heavy duty metal cabinet, switching power supply with rf/line by-passing, 5 inch DS/DD 80 track floppy, Xebec hard disk controller, 25 megabyte winchester hard disk, four serial RS-232 ports and a UNIX C level V compatible multi-tasking, multi-user operating system, the price is under $5000, w/12.5 megahertz system clock (limited time offer). Most all popular high level languages are available at very reasonable cost. The system is expandable to 20 serial ports, at a cost of less than $65 per port, in multiples of 8 port expansion options.

The system SBC fully populated, quality tested, with 4 serial ports pre-wired and board mounted is available for less that $3000. Quantity discounts are available for OEM and special applications, in quantity. All that is required to bring to complete "system" standards is a cabinet, power supply, disks and operating system. All these are available as separate items from DATA-COMP.

A special version of the Motorola 020-BUG is installed on each board. 020-BUG is a ROM based bebugger package with facilities for downloading and executing user programs from a host system. It includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassemble and numerous system diagnostics. Various 020-BUG system routines, such as I/O handlers are available for user programs.

Normal system speed is 3-4.5 MIPS, with burst up to 10 MIPS, at 16.6 megahertz. Intelligent I/O available for some operating systems.

Hands-on "actual experience sessions", before you buy, are available from DATA-COMP. Call or write for additional information or pricing.

DATA-COMP
[illegible] Systems [illegible]
OVER [illegible] YEARS OF DEDICATED QUALITY!

CPI
A Division of
Computer Publishing, Inc.
5900 Cassandra Smith Road
Hixson, Tn 37343
Telephone 615 842-4600
Telex 810 600-6630

Mustang-020 Mustang-08 Benchmarks

```
                                                32 bit     Register
                                                Integer    Long

IBM AT 7300 Xenix Sys 3                           9.7
AT&T 7300 UNIX PC 68010                           7.2        4.3
DEC VAX 11/780 UNIX Berkley 4.2                   3.6        3.2
DEC VAX 11/750      "          "                  5.1        3.2
68008 OS-9 68K 8 Mhz                             18.0        9.0
68000 OS-9 68K 10 Mhz                             6.5        4.0
MUSTANG-08 68008 OS-9 68K 10 Mhz                  9.8        6.3
MUSTANG-020 68020 OS-9 68K 16 Mhz                 2.2        0.88
MUSTANG-020 68020 MC68881 UniFLEX 16 Mhz          1.8        1.22

Main()
     {
            register long i;
            for (i=0; i < 999999; ++i);
     }
Estimated MIPS - MUSTANG-020 ..... 4.5 MIPS.
Burst to 8 - 10 MIPS: Motorola Specs
```

OS-9

OS-9 Professional Ver *Includes C Compiler	$850.00
Basic09	300.00
C Compiler	500.00
68000 Disassembler (w/source add $100.00)	100.00
Fortran 77	750.00
Microware Pascal	500.00
Omegasoft Pascal	900.00
Stylo-Graph	495.00
Stylo-Spell	195.00
Stylo-Merge	175.00
Stylo-Graph-Spell-Merge	695.00
PAT w/C source	229.00
JUST w/C source	79.95
PAT/JUST Combo	249.50
Sculptor (see below)	995.00
COM	125.00

UniFLEX

UniFLEX (68020 ver)	$450.00
Screen Editor	150.00
Sort-Merge	200.00
BASIC/PreCompiler	300.00
C Compiler	350.00
COBOL	750.00
CMODEM w/source	100.00
TMODEM w/source	100.00
X-TALK (see Ad)	99.95
Cross Assembler	50.00
Fortran 77	450.00
Sculptor (see below)	995.00

Standard MUSTANG-020™ shipped 12.5 Mhz.

Add for 16.6 Mhz 68020	375.00
Add for 16.6 Mhz 68881	375.00
Add for 20 Mhz 68020/RAM	750.00
16 Port exp. RS-232 requires 1 or 2 Adapter Cards below.	335.00
RS232 Adapter Each card supports 4 additional ser. ports (total of 36 serial ports supported)	165.00
60 line Parallel I/O card Uses 3 68230 Interface/Timer chips, 6 groups of 8 lines each, separate buffer direction control for each group.	398.00
Prototype Board area for both dip and PGA devices & a pre-wired memory area up to 512K DRAM.	75.00
SBC-AN Interface between the system and ARCNET modified token-passing LAN, fiber optics optional - call	475.00
LAN software drivers	120.00
Expansion for Motorola I/O Channel Modules	$195.00

Special for complete MUSTANG-020™ system buyers - Sculptor $695.00. SAVE $300.00

Software Discounts

All MUSTANG-020™ system and board buyers are entitled to discounts on all listed software: 10-70% depending on item. Call or write for quotes. Discounts apply after the sale as well.

THE PRO!

12.5 Mhz (optional 16.6 Mhz available) MC68020 full 32-bit wide path
32-bit wide data and address buses, non-multiplexed
on chip instruction cache
object code compatible with all 68XXX family processors
enhanced instruction set - math co-processor interface
68881 math hi-speed floating point co-processor (optional)
direct extension of full 68020 instruction set
full support IEEE P754, draft 10.0
transcendental and other scientific math functions
2 Megabyte of SIP RAM (512 x 32 bit organization)
up to 256K bytes of EPROM (64 x 32 bits)
4 Asynchronous serial I/O ports standard
optional to 20 serial ports
standard RS-232 interface
optional network interface
buffered 8 bit parallel port (1/2 MC68230)
Centronics type pinout
expansion connector for I/O devices
16 bit data path
256 byte address space
2 interrupt inputs
clock and control signals
Motorola I/O Channel Modules
time of day clock/calendar w/battery backup
controller for 2, 5 1/4" floppy disk drives
single or double side, single or double density
35 to 80 track selectable (48-96 TPI)
SASI interface
programmable periodic interrupt generator
interrupt rate from micro-seconds to seconds
highly accurate time base (5 PPM)
5 bit sense switch, readable by the CPU
Hardware single-step capability

CPI
Mustang Hi-Speed Systems
Only from Data-Comp Div.
68008-68030

Don't be mislead! ONLY Data-Comp delivers the Super MUSTANG-020

These hi-speed 68020 systems are presently working at NASA, Atomic Energy Commission, Government Agencies as well as Universities, Business, Labs, and other Critical Applications Centers, worldwide, where speed, math crunching and *multi-user, multi-tasking UNIX C level V* compatibility and low cost is a must!

Only the "PRO" version of OS-9 supported!

This is *HEAVY DUTY* Country!

MUSTANG-020 SBC		$2490.00
Cabinet w/switching PS		$299.95
5"-80 track floppy	DS/DD	$269.95
Floppy cable		$39.95
OS-9 68K Professional Ver. * Includes C Compiler ($500.00)		$850.00
Winchester cable		$39.95
Winchester Drive 25 Mbyte		$895.00
Hard Disk controller		$395.00
Shipping USA UPS		$20.00
Total:	Save $1000.00	$5,299.80

complete sytem $4,299.80

UniFLEX	Less	$100.00
MC68881 f/p math processor	Add	$275.00
16.67 Mhz MC68020		$375.00
16.67 Mhz MC68881		$375.00
20 Mhz MC68020 Sys		$750.00

Note all 68881 chips work with 20 Mhz Sys

UPGRADES Write or Call for Professional OS-9 "Full Bore" Upgrade Kit

For a limited time we will offer a $400 trade-in on your old 68XXX SBC. Must be working properly and complete with all software, cables and documentation. Call for more information.

NOTE: Only Professional OS-9 now available (68020 Version) Includes ($500.00) C Compiler 68020 & 68881 supported

complete
25 Mbyte HD System
$4299.80
85 Mbyte HD System
$5748.80

Data-Comp Division
CPI A Decade of Quality Service"
Systems World-Wide
Computer Publishing, Inc. 5900 Cassandra Smith Road
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

Basically OS-9

Dedicated to the serious OS-9 user.
The fastest growing users group world-wide!

6809 - 68030

A Tutorial Series

By: Ron Voigts
2024 Baldwin Court
Glendale Heights, IL

IT DON'T WORK!

Many times I get phone calls from friends who are using OS-9. They have been trying to get some program to run. They have been at the keyboard for a long time with only error messages and a lot of frustration to show for their effort. I ask, "What is wrong?" And the reply is pretty much the same, "IT DON'T WORK!"

If you are an OS-9 user, then this must have happened to you. If you say otherwise, I would believe you are "pulling my leg." Why? Because it happens to me. Now I have been around awhile, long enough to know the basics. But from time to time, things just don't work. This happened the other day.

I tried to run a familiar program and got only an error message. It was error #216 -- File Not Found. It entered DIR X and saw that it was in the file directory. I tried the full path name. Error #215 -- Bad Path Name. Well maybe I can load it into memory. So I try the LOAD command. Yep, you guessed it. It was another error, #214 -- No Permission. Talk about aggravation! I tried using IDENT and got error #243 -- CRC Error.

It was time to play OS-9 detective. I tried DIR E X to gain a little more information about the files in the command directory. It showed me that IDENT's owner was $8C0D, was a directory and contained some impossible number of bytes. In short, it was hurt. Its file descriptor sector appeatred to be glitched. The original program that I tried running had incorrect attributes. I used ATTR on it and salvaged it. Unfortunately, IDENT was a total loss and had to be scrapped. (A little more detective showed that the AC line was the source of the problems. Evidently I was picking up transients. Currently I am trying one of those "spike suppressors". It seems to be dotng the trick.)

So what have we learned from all of this? Specifically, there are a number of things that must be correct for an executable module to run. The module's header check byte, CRC value and attributes must all be correct. If anyone of these is not right, it doesn't get loaded. If it is not loaded, it can't be run.

The module header is the first thing to consider. There are 8 bytes. They are composed of the sync bytes, module size, name offset, type, language, attributes and revision. These 8 bytes are exclusively ORed to create the header check.

Next is the module's CRC. It is located in the last three bytes of the module. This is a check of the entire contents of the module. If so much as a single byte is wrong, this will detect it. It won't tell where it is wrong, but at least it will tell something is wrong.

I won't go into the algorithm of how the CRC is calculated. If you wish to investigate it further, take a look at the system request F$CRC. It can be used to verify an existing module or the CRC can be calculated for a new one.

One other item to mention about the CRC. It is only checked when the module is loaded into memory. Once the module is listed tn the system module directory, it is safe to modify. This can be accomplished with SOLVE from Southeast Media or DEBUG from Microware. I'll show later how to do this.

Finally, there is the attributes of the file. If the module's file does not have execute permission, it will not run. This is what caused the errors 214, 215 and 216, in my case. It is necessary that at least the owner execute be "turned on".

So now we know what goes wrong. How do we attack the problem? If it does not want to run, then there must be a reason. And it is most likely one of the above. (I did not talk about one obvious reason. That is the correct execution directory. Always make certain that you are in the execution directory and that the file is In it. Whenever I change disks, I always do a CHX /DO/CMDS.)

Checking the files attributes is a good idea. The ATTR command is available. Entering it with the modules pathname will generate the necessary information. If you get something like:

```
---wr-wr
```

you most likely have found a problem. So, we will want to correct this. If the executable module's name MYMODULE, we would enter:

```
OS9: attr mymodule -pe -e
```

This would give the module both public and owner execute permission. I usually keep my programs as both, unless I see a reason to exclude others from using them.

The next step, if the attributes are OK, is to check the header check and CRC. This can be done with VERIFY. VERIFY inputs through the standard input path and prints it information to the standard error path. To use it, we enter:

```
OS9: verify <mymodule

Module's header parity is correct.
Calculated CRC matches modules.
```

If come up wrong, the module is probably hurt. But if you believe it is not, then you can use VERIFY to correct it. Let's say MYMODULE is to be corrected.

The U (update) option of VERIFY can be used. It will then copy the module to the standard output path replacing the header check and CRC with the correct values. Entering the following:

```
OS9: verify u <mymodule >newmodule
```

will create a new version that will hopefully run.

These techniques may come in handy sometime. There are no guarantees that a module will run after you have applied them. Generally speaking, if the CRC and header check do not match, the program is hurt. Using VERIFY to correct them won't help. It is best to erase the sick program and replace it with a good one.

FURTHER ANTICS WITH VERIFY

Besides correcting sick modules, there are other reasons why you might want to use VERIFY. If you have to patch a program, it would become necessary to correct the module's CRC. Many times programs come that are not quite the way you expected them. You don't have the source code. You know what you want it to do. The only way to get it there is to patch it.

One such example is the C Compiler licensed to Tandy from Microware. It is hard coded into CCI and C.PREP that the DEFS directory and other items should be found on /D1. Now what if you have only one drive, but it is double sided and called /D0. You have the necessary space for everything, but the C Compiler has its heart on going to drive /D1? You've got to patch it. I'll show you how I did CC1 and let C.PREP up to you.

First, I loaded CCI into memory with the command LOAD. Next I used SOLVE. This is the Symbolic Object/Logic Verify and Examiner debugger form South East Media. I used its monitor commands. I does much more, but I'll tell about it another time. Once into SOLVE, I entered:

```
DBG: L CCI
B000 87 .
```

This linked to CC1. SOLVE told me that CCI started at location $B000. Next I would find where the /D1 is located. I entered:

```
DBG: ? $B000 $D000 "/D1
BEE4
```

This asks, where the string of /D1 is located? And SOLVE said it started at $BEE4. Next I changed it to "...". This would give me a little flexibility. The dots would back me out a few directory levels from where I was at. So if the source code was at /D0/SOURCES, it would use /D0. If I was on the RAM Disk and in /R/SOURCES, it would use /R. So, I entered:

```
DBG: C $BEE4
BEE4 61 / " ...
```

I indicated that I wanted to change the byte at $BEE4. It brought me there and I enterred the 3 dots as a string, using the " indicator. Now all was done. I left SOLVE.

VERIFY could now be used.

```
save /d0/cmds/temp cc1
del -x cc1
verify u <temp >cc1
del -x temp
```

This series of commands saveed the new CC1 as a file called TEMP. It deleteed the old CC1. Then VERIFY was used to correct it. Finally, the file TEMP was eliminated leaving the new version of CC1.

What could be easier? You can use your favorite debugger. I like SOLVE. Once you have changed the portion of the program of interest, use VERIFY to correct the CRC and header check. And you you're all set.

DATE REVISITED

Some time back, I presented a replacement for the command DATE. It appeared in the February '86 issue. It was written in 6809 assembly code. I created it with hopes of improving the original. This version printed time in standard, rather than military notation. It used PM and AM. It also would print the day of the week. For the most part it has worked well. But I have recently have been thinking of changing it. Improving it.

I added a few more criterion to my design considerations. First, I decided to rewrite it in C code. This would make it more universal. Although the final program is a bit longer, I don't think memory is an objection. Second, I added the choice of either standard or military time notation. Finally, it will print the information in a "greeting" type fashion. There are four options that can be used with this new DATE command. They are:

-t = print the time
-m = use military notation
-d = include day of the week
-g = use greeting style

So, if I entered:

OS9: date -t -m -d -g

I might see something like:

Good Evening! Today is July 9, 1987
It is Thursday The time is 18:29:00

This makes a nice addition to your startup file. And it looks a lot more friendly.

Rather than write the program in one large source code, I broke it up into smaller functions. I like to take a modular approach to programming. I don't have to consider a large program chunk, but rather concentrate on smaller discrete units. I took a course one time on programming style. A question was put forth, "How long should a program unit be?" The answer is 60 lines. This is how much space it would take if listed to a printer. It can easily be examined without any page flipping. Actually, longer than this is not necessarily bad. But the longer the program unit gets, more errors can crop up.

Listing 1 is the program DATE. I think it should be fairly understandable. I created my own algorithm for finding the day of the week. One item of interest is the structure time. 'Time' is a pointer to a byte structure. I declared it with

struct sgtbuf *time;

This makes is easy to pass as a parameter. Referencing members of the structure is a little more complicated. For example, the year is referred to as:

(*time).t_year

Since the dot binds tighter, the parentheses are used. This statement says the char value at t_year pointed to by 'time'. Another method to indicate this is:

time->t_year

Either statement will work. I chose to be consistent and stay with one.

I have included another listing this month, LISTING 2. This is a C function I presented last month. It is GETOPT(). I use it to return the options passed on the command line. Actually the program only use a small feature of it. But this does show how to make use of GETOPT(). I plan on using this option parser from time to time. You might want to keep a copy of it somewhere, since I may not always rerun it.

That wraps up another month of BASICALLY OS-9. Take care until next time!

LISTING 1

```
/* *****************************

   Name: Date.c
   Date: 22-Jun-87
   Author: Ron Voigts
   Compiler: Microware C Compiler

   *****************************

   Version 1.00
   Prints the date and time to
   the standard output path.

   *****************************

   Function:
   Prints the date.
   -t = with the time
   -m = time in military notation
   -d = with the day of the week
   -g = with a greeting

    ***************************** */

#define LEVEL2

#include <stdio.h>
#include <time.h>
#include "getopt.c"

#define TRUE 1
#define FALSE 0

/* Parameter flags */
int mflag = FALSE; /* Military time */
int tflag = FALSE; /* Print time */
int dflag = FALSE; /* Print the day */
int gflag = FALSE; /* Greeting flag */
int pmflag = TRUE; /* PM flag */


struct sgtbuf *time;

main( argc, argv )
int argc;
char *argv[];
{

/* Variables used */
  char *option;
  char *optlist="MDTG";

/* Process the input line */
   optn=1;
   while ( (option=getopt( argc, argv, optlist )) != NULL )
      if ( opterr != 0 )
         dhelp();
      else {
         if ( toupper(*option) == 'M' ) mflag = TRUE;
         if ( toupper(*option) == 'D' ) dflag = TRUE;
         if ( toupper(*option) == 'T' ) tflag = TRUE;
         if ( toupper(*option) == 'G' ) gflag = TRUE;
      }

/* Now get the time */
   getime( time );

/* Print the greeting */
   if ( gflag ) {
      if ( (*time).t_hour< 12 )
         printf("Good Morning!  ");
      else if ( (*time).t_hour<18 )
         printf("Good Afternoon!  ");
      else
         printf("Good Evening!  ");
   }

/* Print the date */
   if ( gflag )
      printf("Today is ");
   pdate( time );
/* Print the day of the week */
   if ( dflag ) {
      if ( gflag )
         printf("It is ");
      pday( time );
   }

/* Print the time */
   if ( tflag ) {
      if ( gflag )
         printf("The time is ");
      if ( mflag )
         pmtime( time );
      else
         ptime( time );
   }

}

/* Help for date */
dhelp()
{
   printf("Usage: \n");
   printf("    date [-t] [-m] [-d] [-g] \n");
   printf("        -t = with the time\n");
   printf("        -m = time in military
notation\n");
   printf("        -d = with the day of the
week\n");
   printf("        -g = with a greeting\n\n");
   exit( 0 );
}


/* Print the day of the week */
pday( t )
struct sgtbuf *t;
{

/* Variables used */
   int n;
   register int i;
   int fudge=3; /* My fudge factor */

   static char *day[] = {
      "Sunday",
      "Monday",
      "Tuesday",
      "Wednesday",
      "Thursday",
      "Friday",
      "Saturday"
   };

   static int day_count[] = {
      31, 28, 31, 30,
      31, 30, 31, 31,
      30, 31, 30, 31
   };

/* Calculate today's day of the week */
   n = (*t).t_year + fudge + ((*t).t_year+3)/4;
   for ( i=0; i<(*t).t_month ; i++ )
      n+=day_count[i];
   n+=(*t).t_day;

/* Adjust if this is leap year */
   if ( ((*t).t_year % 4 )==0 && (*t).t_month>2 )
      n++;

/* Print day of the week */
   printf("%s  ", day[ n % 7 ]);

}

/* Print the date */
pdate( t )
struct sgtbuf *t;
{

/* The 12 months */
   static char *month[] = {
      "Unknown",
      "Jaunary",
      "February",
      "March",
      "April",
      "May",
      "June",
      "July",
      "August",
      "September",
      "October",
      "November",
      "December"
   };

    printf("%s %2d, 19%02d\n", month[(*t)
t_month],
             (*t).t_day, (*t).t_year );

}

/* Print the time */
ptime( t )
struct sgtbuf *t;
{

   if ( (*t).t_hour<12 )
      pmflag=FALSE;
   if ( (*t).t_hour>12 )
      (*t).t_hour-=12;
   printf("%2d:%02d:%02d ", (*t).t_hour,
          (*t).t_minute, (*t).t_second );
   if ( pmflag )
      printf("PM\n");
   else
      printf("AM\n");
 }

/* Print in military time */
pmtime( t )
struct sgtbuf *t;
{
   printf("%02d:%02d:%02d\n", (*t).t_hour,
          (*t).t_minute, (*t).t_second );
 }
```

LISTING 2

```
/* ***************************

   Name: GETOPT
   By: Ron Voigts
   Date: 25-MAY-87

   ***************************

   Function:
   This function examines the argument list
   returning a pointer to the option and
   its argument.  A null string is pointed
   to if the option has now argument.

   ***************************

   Version 1.00
   Original.

   *************************** */

#define TRUE 1
#define FALSE 0

char *optarg; /* Option argument */
int optn;     /* Next option     */
int opterr;   /* Error status    */

char *getopt( c, v, optlist )
int c;          /* argument count  */
char **v;       /* argument vector */
char *optlist;  /* option list     */

{
   int isoption;     /* option flag  */
   int hasarg;       /* option argument flag */
   register int i;   /* useful index */
   char *opt;        /* option pointer */
   char *t;          /* argument pointer */
   static char *null='\0'; /* null string */

/* Set up the null string for 'optarg' */
   optarg = null;

/* Set up the error return status */
   opterr = 0; /* No errors */

/* Set up the argument */
   t=v[optn];

/* We are at the end of the argument list */
   if ( (optn==c) || (*t!='-') || ( *t=='-' && *(t+1)=='\0' ) )
      return( 0 );

/* We can set the option */
   opt = t+1;

/* Check if we have an option with an argument */
   isoption = FALSE;
   hasarg = FALSE;
   for ( i=0; i<strlen(optlist); i++ )
      if ( toupper(*(t+1))==toupper(optlist[i]) ) {
         isoption = TRUE;
         if ( optlist[i+1]=='=')
            hasarg = TRUE;
      }

/* If this is not an option then return with error */
   if ( !isoption )
      opterr=-1; /* illegal option */

/* Now we check and set up the argument */
   if ( hasarg ) {
      if (*(t+2) == '\0')
         if ( optn < c-1 )
            optarg = v[++optn];
         else
            opterr=-2; /* Missing option argument */
      else
         optarg = t+2;
      if ( *optarg=='=' )
         optarg++;
   } else
      if ( *(t+2) != '\0' )
         opterr=-3; /* Argument not expected */

/* Now we have an argument and option */
   optn++; /* Adjust the next pointer */
   return( opt ); /* Return the option pointer */

}
```

EOF

The C Programmers Reference Source.
Always Right On Target!

C User Notes

A Tutorial Series

By: Dr. E. M. 'Bud' Pass
1454 Latta Lane N.W.
Conyers, GA 30207
404 483-1717/4570
Computer Systems Consultants

INTRODUCTION

This chapter concludes the discussion of the conversion of Technical Systems Consultants BASIC and Microware BASIC09 programs into C programs begun in an earlier chapter.

CONVERTING BASIC PROGRAMS TO C

Following is a summary of the changes required to convert BASIC function calls to C. Many differences are discussed elsewhere in this series. A brief comment appears with each function.

BASIC FUNCTION		VALUE RETURNED	C EQUIVALENT
ABS	(n)	absolute value of n	iabs(n), labs(n), fabs(n), dabs(n)
ADDR	(n)	address of n	&n
ASC	(s$)	numeric value of first character of s$	*(s)
ASN	(n)	arcsine of n	fasn(n), dasn(n)
ATN	(n)	arctangent of n	fatn(n), datn(n)
CHR$	(n)	ASCII character corresponding to n	((char)(n & 0xff))
COS	(n)	cosine of n	fcos(n), dcos(n)
CVT$%	(s$)	integer equivalent of 2-byte s$	cvtsi(s)
CVT$F	(s$)	floating equivalent of 5/8-byte s$	cvtsf(s)
CVT%$	(n)	2-byte equivalent of integer n	cvtis(n)
CVTF$	(n)	5/8-byte equivalent of floating n	cvtfs(n)
DATE$		current date (may be different format)	dates()
PEEK	(n)	16-bit numeric value at address n	((*((char *)(n)) << 8) | (*((char *)(n + 1)))
EOF	(#n)	test if path n at end of file	none
ERL		error line number	none

```
ERR                                errno
               error number (will be different)
EXP      (n)                       fexp(n), dexp(n)
               e to power n
FALSE                              0
               false boolean constant
FIX      (n)                       fint(n+0.5), dint(n+0.5)
               rounded integer value of n
FLOAT    (n)                       ((float)(n)), ((double)(n))
               n converted to floating point
FRE      (n)                       none
               number of bytes of unallocated memory
HEX      (s$)                      hex(s)
               numeric equivalent of hex string s$
INCH$    (n)                       inchs(n)
               input one character from file n
INSTR    (n,s1$,s2$)               strchrs(s1 + (n),*s2)
               first occurrence of s2$ in s1$ starting
               with character n or zero if not found
INT      (n)                       fint(n), dint(n)
               truncated integer value of n
LAND     (m,n)                     ((m) & (n))
               bitwise m and n
LEFT$    (s$,n)                    mids(s,1,n)
               string representing n characters
               at beginning of s$
LEN      (s$)                      strlen(s)
               length in bytes of s$
LNOT     (n)                       (~(n))
               bitwise not n
LOG      (n)                       flog(n), dlog(n)
               natural logarithm of n
LOG10    (n)                       flog10(n), dlog10(n)
               base 10 logarithm of n
LOR      (m,n)                     ((m) | (n))
               bitwise m or n
LXOR     (m,n)                     ((m) ^ (n))
               bitwise m exor n
MID$     (s$,n1,n2)                mids(s,n1,n2)
               string representing n2
               characters starting at n1
               characters into s$
MOD      (m,n)                     ((m) % (n))
               remainder of m divided by n
PEEK     (n)                       *((char *)(n))
               8-bit numeric value at address n
PI                                 3.14159265
               pi (3.14159265)
```

```
POS      [(n)]                     none
               character position in file n buffer
PTR      (v$[(n[,n[,n]])])         &vs, &vs[n], &vs[n][n], &vs[n][n][n]
               address of argument (see text)
PTR      (v[(n[,n[,n)])])          &v, &v[n], &v[n][n], &v[n][n][n]
               address of argument (see text)
RIGHT$   (s$,n)                    rights(s,n)
               string representing n characters at
               end of s$
RND      (n)                       rnd(n)
               random number between 0 and 1
SGN      (n)                       isgn(n), lsgn(n), fsgn(n), dsgn(n)
               sign of n
SIN      (n)                       fsin(n), dcos(n)
               sine of n
SIZE     (n)                       sizeof(n)
               size of n
SPC      (n)                       lefts(spaces,n)
               generate string of n spaces
SQ       (n)                       ((n) * (n))
               n squared
SQR      (n)                       fsqr(n), dsqr(n)
               square root of n
SQRT     (n)                       fsqr(n), dsqr(n)
               square root of n
STR$     (n)                       strs(n)
               string conversion of numeric n
SUBSTR   (s1$,s2$)                 strchrs(s2 + n,*s1)
               first occurrence of s1$ in s2$
               or zero if not found
TAB      (n)                       none
               advance print buffer pointer to
               position n
TAN      (n)                       ftan(n), dtan(n)
               tangent of n
TRIM$    (s$)                      trims(s)
               string conversion of numeric n
TRUE                               1
               true boolean constant
USR      (n)                       none
               user function with parameter n

VAL      (s$)                      val(s)
               numeric conversion of string s$
```

Statements

Statementa provide the means of specifying the operations to be performed and the order in which they are to be performed in the BASIC language. The statements supported by the TSC BASICs and by BASIC09 are similar, although not identical. Many statements have already been discussed in previous sections of this discussion. Several of the more important ones are discussed below.

Compound statements are separated by ':' and '\' in the BASIC interpreters and by ';' in the C compiler. Complex statement groups are formed in BASIC by placing them on the same line, separated by ':' or '\'. They are formed in C by surrounding them with '{' and '}'.

The IF statement has somewhat different formats among the TSC BASIC and BASIC09 interpreters and in the C compiler. The primary effect this has on the conversion is that an open brace must be placed before and a close brace after each group of compound statements between the THEN and the ELSE or end of the logical line and between the ELSE and the end of the logica line.

A subtle difference between the TSC BASIC FOR statement and the BASIC09 FOR and C for statements concerns the fact that the TSC BASIC FOR structure always executes the body of the structure once, whereas the BASIC09 and C structures check the terminating condition first, before executing the body of the structure initially. Also, because of the different methods used by TSC BASIC and BASIC09, multiple NEXT v statements for the same FOR statement, although legal in TSC BASIC, may generate different results in BASIC09 from those generated in TSC BASIC, and, of course, there is only one exit point from a C for statement.

The BASIC09 PROCEDURE declaration statement may be conversion into a C function header. Any PARAM declarations may be converted into C function arguments, with the considerations discussed in an earlier chapter. Any DIM and other declarations may be converted into local or global variables, as appropriate.

The GOTO statement of BASIC and the goto statement of C perform essentially the same function. If labels are preceded by 'L_', or some other unique prefix, numeric BASIC labels will be made into legal C labels and alpha labels happening to coincide with C reserved words will also be made into legal C labels. Duplicate label identifiers in separate BASIC09 PROCEDURES could be made unique by prefixing each group with unique strings.

One consideration which has the capacity of materially complicating the conversion of BASIC programs into C programs is the run-time structure imposed by the GOSUB statement. If a BASIC program is written in a modular fashion such that there are no or very few branches into and out of GOSUB subroutines, the conversion is far simpler than in the case of the traditional "bowl of spaghetti" BASIC programs in which a large number of GOTO and GOSUB statements are freely intermingled. The primary conflict arises from the consideration that the BASIC GOSUB statement should ideally be converted into a C function call and the target label should be converted into a C function header. Branches into and out of a C function are always illegal, even though many C compilers may not catch the error. In extreme cases, it may be necessary to rewrite and retest the BASIC code before conversion.

The TSC BASIC DIGITS statement has no direct equivalent in C. If the formatting changes caused by the DIGITS statement are important, some of them may be recovered thru the use of the appropriate formatting options of the C fprintf statement. This includes the precision and width options of the C printf statement.

The TSC BASIC FIELD statement and the BASIC09 TYPE statement are often used to structure records to be read and written from and to disk files. Other than the potentially dynamic status of the FIELD statement, both statements may be converted to C structures. However, very few individual FIELD statements or TYPE statements define entire records, so it is necessary to somehow map out the record format or formats of interest in order to determine the corresponding C structure.

The GET and PUT statements of both versions of BASIC perform the actual input and output into and from these BASIC structures. In TSC BASIC, the optional record number designations allow random record access; since the C fseek function requires the offset to be provided in bytes, not records, the corresponding fseek argument must be multiplied by the size of the structure.

DATA statements in TSC BASIC are considered as constants, and no evaluation is performed. DATA statements in BASIC09 are considered as expressions, which are evaluated as they are encountered thru the corresponding READ statementa. The RESTORE statement resets the point of evaluation of the DATA statements. All TSC BASIC sequences intended to be processed as character strings must be enclosed in double quote characters. Depending upon the logic of the program, the DATA statement may be converted as a C initializer or as a series of assignments.

The C language has no single statement directly corresponding to the TSC BASIC statement SWAP, which exchanges its two arguments. However, the SWAP statement may always be expanded into three BASIC statements, in the following manner:

```
     SWAP v1$,v2$      temp$=v1$\
                        v1$=v2$\
                        v2$=temp$

      SWAP v1,v2        temp=v1\
                        v1=v2\
                        v2=temp
```

thus eliminating the SWAP statement entirely.

Following is a summary of the changes required to convert TSC BASIC and BASIC09 statements to C statements. A brief comment appears with each statement conversion. When necessary, T indicates TSC BASIC and B indicates BASIC09.

BASIC STATEMENT	OPERATION PERFORMED	C EQUIVALENT
BASE N	set index base	none
BYE	terminate execution	exit (0)
T CHAIN s$ [N]	load and run BASIC program named s$ starting with line N or first line	system (s)

```
B CHAIN    s$                            system (s)
                execute command line s$
CHD        s$                            chdir (s)
                change current data directory
CHX        s$                            none
                change current execution directory
T CLOSE    n[,n[,...]]                   fclose (n)
B CLOSE    #n[,#n[,...]]
                close specified files n1, n2, ...
CREATE     #,s$:[WRITE][+UPDATE][+EXEC]  n = fopen (s, "w")
                open new file n1 with name s$
DATA       N[,N[,...]]                   initializers, assignments
DATA       S$[,S$[,...]]
DATA       "S$"[,"S$"[,...]]
                establish constant table within program
DEF        FNv(x)=n                      #define FNv(x) (n)
                define numeric function FNv with parameter x
DEG                                      none
                change trig functions to degrees
DIGITS     n[,n]                         none
                set number of digits to be printed
B DIM      v(n[,n])[,...]:BOOLEAN        char v[n] ...
                declare dimensioned boolean values
B DIM      v(n[,n])[,...]:BYTE           char v[n] ...
                declare dimensioned characters
T DIM      [#n,]v$(n[,n])[,...][=N]      char vs[n] ...
                declare dimensioned strings or declare
                virtual array associated with file n1
B DIM      v$(n[,n])[,...]:STRING[N]     char vs[n] ...
                declare dimensioned strings
T DIM      [#n,]v%(n[,n])[,...]          int v[n] ...
                declare dimensioned integers or declare
                virtual array associated with file n1
B DIM      v(n[,n])[,...]:INTEGER        int v[n] ...
                declare dimensioned integers
T DIM      [#n,]v(n[,n])[,...]           int v[n] ...
                declare dimensioned floating point variables
                or virtual array associated with file n1
B DIM      v(n[,n])[,...]:REAL           int v[n] ...
                declare dimensioned floating point variables
DPOKE      n1,n2                         {*((char *)(n1)) = (n2 >> 8);
                                         *((char *)(n1 + 1)) = n2}
                store 16-bit value n2 at address n1
END                                      return
                terminate PROCEDURE or program
END        s$                            {printf ("%s\n", s);
                                         return; }
                print string and terminate PROCEDURE
```

```
ERROR      (n)                           none
                generate error n
EXEC       ,s$                           system (s)
                send command line s$ to operating system
EXITIF     x THEN S1 ... ENDEXIT         if (x) {s1; ...; break; }
                if expression true, execute statements and exit
FIELD      #n,n AS v$[(n[,n])][,...]     (establish structure)
                establish fields in random I/O buffer
FOR        v=n1 TO n2 [STEP n3] ... NEXT v (see text)
                create loop structure with control variable
                v set initially to n1, terminal condition
                of v crossing n2, step size n3 (or 1)
                (C and BASIC09 check condition before first time,
                TSC BASIC always executes body once)
T GET      #n1[,RECORD n2]               (see text)
                read random file n1 record n2 or next
B GET      #n1,structure                 (see text)
                read file n1 record into structure
GOSUB      N                             (see text)
                call subroutine starting at label N
GOTO       N                             goto L_N
                branch to label N
IF         x GOTO N                      if (x) goto L_N
                branch to label N if expression x true
IF         x THEN N1                     if (x) goto L_N1
                branch to label N1 if expression x true
IF         x THEN N1 ELSE N2             if (x) goto L_N1;
                                         else goto L_N2
                branch to label N1 if expression x true;
                otherwise branch to label N2
IF         x THEN S1                     if (x) then {s1}
                perform statement S1 if expression x true
IF         x THEN S1 ELSE S2             if (x) then {s1};
                                         else {s2}
                perform statement S1 if expression x true;
                otherwise perform statement S2
B IF       x THEN S1 [ELSE S2] ENDIF     (same as IF)
                same as IF above
INPUT      LINE [#n,]vS                  fgets (vs, size, n)
                input string v$ from file n or terminal
INPUT      [#n,]L                        {fprintf (n, "%s ");
                                         fscanf (n, "...", L);
                input list L from file n or terminal
INPUT      [#n,][S$;]L                   {fprintf (n, "%s ");
                                         fscanf (n, "...", L);
INPUT      [#n,][S$,]L                   {fprintf (n, "%s ");
                                         fscanf (n, "...", L);
                issue prompt S$ to file n or terminal
                and input list L from file n or terminal
```

```
T KILL    s$                           unlink (s)
                delete file named s$
B KILL    s$                           none
                unlink PROCEDURE
[LET]     v$[(n[,n])]=s$               same
                assign expression on right of equal
                to variable on left (see text for string
                and virtual array conversions)
[LET]     v[(n[,n])]=n                 same
                assign expression on right of equal
                to variable on left
                (see text for virtual array conversions)
LSET      v$[(n[,n])]=s$               (same as LET)
                assign expression on right of equal
                to field variable on left of equal
                in left-justified mode
LOOP      S1 ... ENDLOOP               while (1) {s1; ... }
                perform statements endlessly
NEXT      v                            (end of for structure)
                initiate next iteration for FOR loop
                with control variable v
ON        ERROR GOTO [N]               signal (...)
                set trap at label N for error interception
ON        ERROR GOTO [0]               signal (...)
                terminate error interception trap
ON        n GOSUB N1[,N2[,...]]        switch (n)
                                       {
                                       case 1:
                                           goto L_N1;
                                       case 2:
                                           goto L_N2;
                                       :
                                       }
                call subroutine at n-th label N
ON        n GOTO N1[,N2[,...]]         switch (n)
                                       {
                                       case 1:
                                           L_N1();
                                           break;
                                       case 2:
                                           L_N2();
                                           break;
                                       :
                                       }
                branch to n-th label number N
OPEN      #,s$:[READ][+WRITE][+UPDATE][+EXEC][+DIR] n = fopen (s, "r")
                open old file n1 with name s$
OPEN      NEW s$ AS n                  n = fopen (s, "w")
                open new file n1 with name s$
```

```
OPEN      OLD s$ AS n                  n = fopen (s, "r")
                open file n1 with name s$
OPEN      s$ AS n                      n = fopen (s, "r+")
                open file n1 with name s$
PARAM     ...                          (like DIM)
                declare PROCEDURE arguments
PAUSE                                  exit (0)
                pause and enter diagnostic mode
PAUSE     s$                           {printf ("%s\n", s);
                                       exit (0); ]
                print string, pause and enter diagnostic mode
POKE      n1,n2                        *((char *)(n1) = (n2)
                store 8-bit-value n2 at address n1
PRINT     [#n[,]][USING s$[,]]{e[,]}{e[;]}[,...] (see text)
                output characters to file n or terminal
PROCEDURE N                            (see text)
                start PROCEDURE declaration
T PUT     #n1[,RECORU n2]              (see text)
                write random file n1 record n2 or next
B PUT     #n1,structure                (see text)
                write file n1 record from structure
RAD                                    none
                change trig functions to radians
READ      L                            (see text for DATA)
                read data into list L from DATA statements
READ      #n,L                         none
                read null-delimited list from file n
REM       ...                          /* ... */
*         ...                          /* ... */
                introduce remark
RENAME    s1$,s2$                      none (o-s dependent)
                rename file named s1$ as s2$
REPEAT    S1 ... UNTIL x               do {s1; ... } while (!(x))
                perform statements unitl expression true
RESTORE   [N]                          (see text for DATA)
                reset DATA pointer to first statement
                or to statement at label N
RESUME    [N]                          (see text)
                return to program from error routine
                to original statement or to label N
RETURN                                 return
                return from most recent active GOSUB
RSET      v${(n{,n})}=s$               (same as LET)
                assign expression on right of equal
                to field variable on left of equal
                (in right-justified mode)
RUN       procedure [(p1 [, ...] )]    procedure ([p1 [, ...] ])
                call procedure with arguments p1, ...
SEEK      #n,m                         fseek (n, (long)m, 0)
                position file n to byte m
```

```
SHELL     s$                          system (s)
                pass command string to operating system
STOP                                  exit (0)
                terminate program
STOP      s$                          {printf ("%s\n", s);
                                      exit (0); }
                print string and terminate program
SWAP      v$[(n[,n])],v$[(n[,n])]     (see text)
                exchange contents of specified variables
SWAP      v[(n[,n])],v[(n[,n])]       (see text)
                exchange contents of specified variables
TROFF     ...                         none
                turn trace off
TRON      ...                         none
                turn trace on
TYPE      ...                         none
                declare new variable types
WHILE     x DO S1 ... ENDWHILE        while (x) {s1; ... }
                while expression true perform statements
WRITE     #n,L                        none
                write null-delimited list to file n
```

EXAMPLE C EXAMPLE

Following is this month's example C program; it provides a version of the echo program compatible with both UNIX System 5 and with UNIX BSD 4.2.

```
#include <stdio.h>
#include <ctype.h>
#include "version.h"

char string[256], *expand(), *p, *q;
int c, i, j, nl, sp;

main(argc, argv)
int argc;
char *argv[];
{
    for (i = 1; i < argc; i++)
        fputs(expand(argv[i]), stdout);
    if (!nl)
        putc('\n', stdout);
    fflush(stdout);
    exit(0);
}

char *expand(a)
char *a;
{
    if (!strcmp(a, "-n"))
    {
        ++nl;
        return "";
    }
    p = string;
    if (sp++)
        *p++ = ' ';
    while (*a)
        if ((*p++ = *a++) == '\\')
        {
            switch (c = *a++)
            {
            case 0:
                --a;
                break;
            case '0':
                for (c = j = 0; isdigit(*a) && (j < 4);
                    ++j, ++a)
                    c = (c << 3) + (*a - '0');
                *(p - 1) = c;
                break;
            case 'b':
                *(p - 1) = '\b';
                break;
            case 'c':
                --p;
                ++nl;
                break;
            case 'f':
                *(p - 1) = '\f';
                break;
            case 'n':
                *(p - 1) = '\n';
                break;
            case 'r':
                *(p - 1) = '\r';
                break;
            case 't':
                *(p - 1) = '\t';
                break;
            case 'v':
                *(p - 1) = '\v';
            case '\\':
                break;
            default:
                *p++ = c;
            }
        }
    *p = 0;
    return string;
}
```

EOF

SOFTWARE USER NOTES

A Tutorial Series

By: Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, MI 48105

From Basic Assembler to HLL's

OS-9, PLuS and PAT

Today is Saturday. Last Monday I decided to bite the bullet and start to do a version of PAT for PLuS. I had dumped the FLEX 6809 PL/9 source code over to the Mustang system and made a few starts at the conversion. Primarily, the difference is in how OS-9 handles files. Actually OS-9 takes care of more of the detail for the user than FLEX does. I went through the code removing the File Control Block declarations and putting filenames into strings to be pointed at when calling OS-9 to open files. On opening a file, OS-9 returns a path number used from that point on to refer to the file, including when it is closed.

The PLuS compiler files.lib did not have the equivalent of the GETFIL call in FLEX to get a filename from the command line, but with a little help from the OS-9 manual I was able to write one, as I mentioned last time. Anyway, I got down to business, first getting the whole big file to compile without errors. On Monday night, I had reached that point, but the compiled output didn't run at all. It just didn't do anything. I had to reset the computer to get control back. (Fairly normal for translating a big program). The following week consisted of long evenings of finding and fixing bugs, but after that time, the job was about done.

One problem that I had was that in FLEX a file error is returned in a location in the File Control Block, so for example, when end of file is reached in reading an input file, an 8 is stored in the second location in the FCB. Later I could and did go back and look at the FCB to see if the input file had run out. OS-9 returns an error in a register after a file operation, and so the end of file error on the last read of an input file goes away on the next file operation. I had to save the fact that the input file had been emptied in a flag I called "eo_infile". I had the opposite problem in a few instances where I carelessly got operations out of order in a loop and tested the error location before doing the next file operation. If a file error was left in the location called OS-9_error in the PLuS library, I had an erroneous error message. The order had been OK in the FLEX version because no error would be present before performing a file operation on a new FCB. I finally got the operations and the tests in the right order, and things were better from that point.

I learned how to ask OS-9 for a big buffer space and set PPAT as I have called the Plus version for now (to distinguish it from the C PAT that I was using to edit it), to use a 100K buffer so I could use it to edit itself. I still had a few peculiar file error messages under some conditions, but they did no harm. A couple days later, I spent the evening trying to find a bug that caused the output file to be open after I thought I had closed it, so I couldn't delete the file on abandoning an edit. It turned out, of course, that I had misunderstood the CREATE procedure which not only creates a directory entry, but also opens a file. I had opened it the second time (on another path) by opening it for write. I am still puzzled as to why OS-9 didn't object to opening the same file twice on two different path numbers. With that cleared up, the file handling seemed to be all squared away.

The compile time for the whole PPAT on the Mustang 68020 system was 22 seconds. On the 68008 system it was 70 seconds. I found one place where the PLuS code is slower than the Microware "C" version, in the loading and saving of files from and to the hard disk. The PLuS version took about three times as long to read a file as the "C" version. In all other respects it seemed to be about equivalent in performance, and PLuS generates about 20% less object code.

I wrote a letter to Windrush to inquire about the slow file reading, but I had the thought that perhaps "C" had a buffer built in as part of the "C" runtime package, so it could read a block of data from a file and then get the data via a shorter and quicker program path. I noticed that PLuS files.lib had a "read_n" procedure. I had used the single character read call "read(path)" which returns one character. The file was input by reading it one character at a time into the edit buffer until an end of file error was detected. PPAT loaded its own source, about 65K of text, in 68 seconds. I changed the procedure to read a large block at a time using "read_n(infile,10000,.buffer);" That call essentially read 10000 characters at a time. Then I could increment the pointer by the number returned by the procedure, which is the actual number of characters read. It will read 10000, the number specified, unless the end of file is reached. On end of file it stops reading, and the procedure returns the number of characters actually read. I found that it read one character past the end of the file, so I had to decrement my pointer, and all was well. There was a slight improvement. Now it reads the source file in just about 2 seconds, about 35 times faster. Clearly there is a large overhead in the OS-9 call to read a single character. About then I had the thought of telling read_n to read 100000 characters since the parameter is a LONG data type, and that it would stop at the end of file and return the number read. I tried it and got into difficulty with finding garbage past the end of my input file. I don't understand why it didn't work but I backed up to the previous version and all was fine again.

Writing the buffer to the output file was even easier. I know ahead of time just how many characters the file contains. It is first moved to the beginning of the buffer, so that it starts at location (0) of the buffer. The pointer 'below' then contains the count of characters. OS-9 and the files.lib of PLuS also support a "write_n" procedure, so I simply wrote all the characters out in one call. The file was written in just about three seconds! I note with some puzzlement, that the 68008 system actually reads and writes this long file from and to the hard disk faster than the Mustang 68020 system. There are two possible reasons for this. The Mustang contains a Xybec disk controller and the

Peripheral Technology 68008 board is connected to a Western Digital controller. Secondly, the P.T. system is running the newest version "Professional OS-9" while the Mustang is running OS-9 of a year ago. Perhaps the file accessing has been streamlined. The difference in performance is slight, but certainly obvious. (Don Williams has confirmed that the difference is because of the disk controllers).

The systems perform quite well. PPAT will move from the top of this 65K file to the bottom in about three seconds on the 68008 system and about twice as fast on the 68020. The most visible difference was in a processor intense procedure of searching for a string. I put a unique string at the end of the big 65K file and search for it from the top of the file. The 68008 version took 5 seconds to find the string and 2 more to move to it. On the 68020 system, the search seemed to be more like a second, with one more to move to the bottom of the file. The time does not seem to depend on the length of the string. I suppose a non-match is generally detected after comparing one or two characters. These times were after a little optimizing of the search routine.

I figured at the start of the project that a 50 page source code program would be a pretty good test for PLuS. It was more of a test of my programming ability and understanding in the area of the file handling, since I found only one new bug in PLuS, and I am not yet sure that it is an error. The manual doesn't mention the MOD function, though the old PL9 has that function implemented. Anyway, the compiler didn't complain when I used that function, but it didn't give me the correct results. I found no other bug in PLuS all week.

Later I decided to look at the string search procedure in a little more depth. I spent an hour flow charting the 20 or so line procedure and thought I had really streamlined it. Later I typed it in and tried it, finding no detectable improvement over the previous versions. I thought of several possible improved ways to do the search. One idea was to search for a match of an integer or long value, by fooling the compiler into thinking the search string and buffer were arrays of integers or longs. I suspect that with a lot of fooling around with memory alignment and careful handling of the start and end of the buffer, that approach would yield a faster search, but I have to look at each character in the buffer and determine if it is a CR. I must keep track of the current line, so when I see a CR I must increment or decrement the current line variable. This would mean that my new scheme would involve scanning the buffer twice, and so would probably slow things down.

The point is that eventually one reaches the point of diminishing returns in trying to improve a program's execution time. At this point, I could devise a new method of keeping track of line numbers, perhaps by imbedding line numbers in the buffer. Then I could just match the string and go look for the line number at the beginning of the current line. An integer double byte would do for line numbers up to 32767, and I would then have to add some distinguishing character to flag a line number.

To my never ending list of possible improvements, I add the idea of storing the text in the large edit buffer in compressed form. That is, wherever there are two or more spaces in sequence, use the HT $09 code followed by a space count. The loading of the large edit buffer to the screen buffer would expand these horizontal tab functions and vice versa. This would mean that one could edit a much larger program in one chunk, and programs could be stored in significantly smaller files. The PLuS source file for PPAT is about 65K characters. There is a great deal of indentation in the program source. Many lines reach an indent value of 15 or more, and most all lines start with at least three spaces, so there would be considerable compression.

Impressions

I have been working with the 68008 and the 68020 systems rather extensively for the past couple of weeks, and for several hours day, at least for last week, the 2 MHz 6809 system with 8" floppy disks. Let me say that after several days with the 68008 system running in the same room with the 6809 system, it is a real drag to go back to that 6809 system. I was compiling programs of about equal size (22-23K of object code) on the two systems. First, after the optimization described above, the file load and save operations for editing got to be intolerable on the old system. Then I noted that I could compile a program three times on the 68008 system while the 6809 system was still chugging along compiling a program. The speed advantage for processor intensive operations seems to be right around 3.5.

Though OS-9 is a fine operating system, there are applications in which it tends to get in the way. In our use of computers in our company's product, we now are set up to plug a disk drive and terminal into the final hardware that is going to be shipped as a special measuring system and machine control combined. We can then develop the program or at least debug it, right in the final hardware, recompiling and trying the code immediately. When we are satisfied with the overall performance, we go program some EPROMs and the job is done. Obviously for our purposes, something simpler like SK*DOS would be more straightforward. Our final product runs without operating system overhead of any kind, so the simpler the O.S. the easier to "disentangle" the program from it. SK*DOS looks very promising for our future, and I have been doing some preliminary things with it, though we are hopeful for a bit more in the way of operable software to be available for SK*DOS before we can really begin to make the switch. I've promised to do a version of PAT for SK*DOS, hopefully when the present efforts to get a fairly full "C" compiler come to fruition. It has occurred to me that I could write the I/O and File Handling libraries for PLuS so that its output code would run under SK*DOS, but then I would have to compile under OS-9, convert to the S1-S9 format, run the code out a serial port to another system, switch operating systems to SK*DOS, read the code back in from the other external system, and convert it to a true binary loadable file. While that cycle is not unusable once the library files are debugged, I wouldn't want to have to debug communications programs, conversion programs, I/O and FILE handling libraries AND my 50 page PAT program all at the same time. Neither do I have the luxury of a great deal of time to spend working out so many phases of the project. Given a reasonable "C" compiler I can start with my "C" version that now runs under OS-9, use the serial trick to get the source file on the hard disk under SK*DOS, edit with the very usable editor that is available, and get PAT working on that system.

Folks at Windrush, if this seems like unfair pressure, it is, and I hope it works. You've told me you have no intention of getting PLuS up under SK*DOS. I told you, and I haven't changed my mind, that PLuS to run under SK*DOS would probably do more for SK*DOS than it would for PLuS, so I can see why you might not be willing to expend the effort to do the job. Let me jut say that I see the combination as the only way our company can go, and that doing so would put us in good shape in our plan to "upgrade" our product to 68XXX processors for the next several years. Graham Trott tells me that PLuS is written in PLuS, and that I would have an easier time following it than the original code in assembler for the 6809. I publicly volunteer to give a good try to porting PLuS over to the SK*DOS operating system if you folks will let me have the source or the minimum part of the source that I need in order to write and debug the links to SK*DOS I/O and disk file handling. In exchange, I will agree to make the result your property to sell. I don't want any royalty. Frankly I don't know if I am up to the task, but I feel the need so strongly for an efficient compiler and a simple operating system combination that I am willing to give it a try. I might also mention that I might need more than a little help from Peter Stark as well, though the SK*DOS manual is fairly extensive and descriptive in terms that I understand.

A couple of weeks have gone by since the writing of the above, and I have managed to get PAT in PLuS form finished to the point of reading the same terminal configuration file as the "C" version. The PLuS version is still smaller than the "C" version by about 10%, which is not really very significant. I've added one more parameter to the terminal configuration files for both versions, the size of the edit buffer in thousands of bytes. I got tired of hitting ESC on the OS-9 system and popping out of my current shell and therefore losing my environment parameters. I had one other thought that would eliminate the environment parameters altogether. PAT could look for the terminal configuration file 'termcon' in the current working directory. The user would have to have a copy in each directory in which he edits, but on a multi-user system, each user could have a different version of termcon (given that each user has a different terminal, of course) in his own working directories. (I had even thought briefly of having the path specified in the termcon file, but having to find the file to read the path to it wouldn't work at all, so I gave that one up as my dumb idea of the week).

Programming Snarl

Have you ever noticed that the hardest part of getting most programs running is the control of the program flow? That is, in a language like Pascal or "C" (or PLuS) the careful construction of the IF-THEN-ELSE structures. Sometimes the scope of a THEN or and ELSE can become obscure, particularly if each is a long compound statement with a number of BEGIN - END pairs enclosing WHILE or other IF THEN statements. I spent all afternoon today trying to untangle a single page of code. Of course I was looking at the wrong part of the code to find the error. I had set up an array of strings and was finding a particular search string in the array, and then associating its position with an index into other arrays to store or recall information. The string was acrually an 8 digit part number. I blamed my string search procedure for most of the afternoon, but then by adding a couple of print statements I found that I was never storing the string in the array in the first place. At that point I quickly found that an ELSE followed the wrong END. Moving it to the correct place gave me instant results and the program now runs fine. The problem would have been obvious except for the fact that my indenting had been fouled up by a program change previously.

My first program for a balancing machine with my current company all fit in one 2716 EPROM, and it used a 6116 2K by 8 RAM. The whole program (in assembler) and RAM fit on a SWTPc 6809 processor board. This latest program will use 24K of ROM in the form of 2764As, and two 8K by 8 battery backed up RAM chips for a total of very nearly 24K of program and 16K of RAM. In some other and larger applications we have used multiple processors, or rather more like multiple computers communicating via serial ASCII data. We use software handshaking. That is, each command is acknowledged with an appropriate and different response or an error status message. We've come a long way since that first 2K program (I had about 20 bytes free in the 2K ROM). It becomes obvious at this point that we are about to outgrow a 64K memory map with our main programs. We figure that the 68008 will give us a 2.5 to 3 times speed advantage over a 2 MHz 6809, and what for now will surely seem like "unlimited" memory for program and data.

Reminiscences

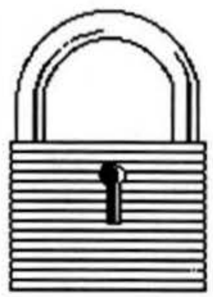
I was thinking the other day about my first little computer system other than that initial single board computer with the LED display and the Hexadecimal keypad. If I remember correctly, one memory board in that system held 4K bytes. It was available as a 4K board with 2K worth of chips and a 2K expansion kit. I bought that first system with 8K BASIC and 16K of RAM, and on that, I had my first taste of programming in BASIC. That 16K cost around $400. Later I bought some 8K boards at around $225 each, making 64K of memory about $1800, but then the power requirements were such that one would have to rebuild the power supply in those early SWTPc boxes to power that much memory. Also there were only 8 card slots and 56K of memory possible because of the monitor and I/O addressing. I understand that now I can buy an 8K by 8 CMOS RAM chip for about $15, so I could have my 64K of STATIC RAM for $120. Of course now, anything less than 512K or 1 Mbyte is considered "small".

Am I saying I want to go back to "the good old days"? No way! I had BASIC, a good assembler, and a dumb line editor to use. I couldn't do very much very fast with that sort of system. My first programming was done with cassette tape for mass storage. That 8K BASIC took a good 5 minutes to load to memory. I would start it loading and go to eat supper, returning to find it ready to run, or having stopped with a read error. A year later I bought a pair of 5" floppy disk drives and wondered how I would ever fill a disk with files. (The old SSSD 35 track disks held some 85K bytes or so of files). When the DSDD 8" disk drives got to the point of being feasible to purchase, I bought a couple and realized that each disk held about as much as 12 of the little disks, though those disks never were filled to capacity so in practice I put about 15 disks worth of information on one 8" disk. Now it is clear that every bit of software that I run on this old 6809 system, and all my data files would easily fit on one 20Mbyte hard disk.

No, I don't want to go back. Bring on the latest and newest in faster microprocessors, hard disks, operating systems, user software... I'll be right here waiting to use them. Hopefully I will be able to write about them too.

EOF

PASSWORD



A SECURE INVESTMENT

"Mine..." Eyebrows raised, I turned and looked at my son. He was sitting where he could see the screen. "That's your password, isn't it?" Even though, it wasn't displayed, he had guessed it. It gave me pause to think. I have worked with computers, for years. Most of them, multiuser systems. I had always prided myself on being security conscience. Why was I using a simple four letter password on my home system?

I changed my password right away after that, but it made me think about another incident. It happened sometime ago with the same son. Now this kid is pretty bright, but I think, no smarter than any other who had been around home computers since he was four years old. Steven needed to transfer an OS-9* file from a standard format disk to one for his CoCo*. Even though, he had utilities on his own system to do it, he went with the easier way of using my "/cc0" (for COCO*) device descriptor and compatible driver software. No problem. Except that the file belonged to user zero, and on my system he has a different user number. Well, he signed off and tried signing on as superuser using the same password as was on his own system. It worked. The system manager password hadn't been changed since I bought the computer!

These two incidents spun around in my mind for a while before I decided that part of the problem was that it was hard to change passwords on an OS-9* system. Well, not HARD, but inconvenient. I had to sign on as superuser and edit the password file and then go back and sign on as myself to go about my business. There was no user accessible password changing program. I looked around, but couldn't turn one up that I liked. So I wrote my own. My feeling is that even for a person who works with computers all the time system security begins at home.

What sort of program did I want? I have always worked with multiuser systems as part of my job. I have come to expect a certain format from password changing programs. Basically they make sure you are who you say you are and then ask you for your new password and change the password file to reflect your request. I didn't want to take forever to write the thing. I like the TOOLS approach to systems. I would rather use a group of little programs (especially if they are already written) than one big one to accomplish something. Finally, I still have my trusty (rusty?) level one system and memory is a concern. All of these constraints can easily be handled by BASIC09*.

The program comes in two parts. One part is an assembler subroutine called USERID. In order to change the password file, the program has to have access to it. User zero is usually the owner of the file and should be the only user who can read it. After all what does all this accomplish if everybody can read it? There are two ways to get at a file you don't own. You can change the attribute bytes in the file header or you can make the system think you are the legal owner. Going after the file header is fairly messy and the program still has to know who you are so it doesn't change someone else's password for you. User zero should be the only one with that kind of power. Even If you know someone elses userid and password, you should still have to go to the trouble of signing on as him to muck up his password. Now since we need to know who called us and we need to have access to a file that probably doesn't belong to us, we need a way to get to the descriptor for our current process. OS-9* has such a way. It comes as two OS-9* system calls. These are F$id and F$suser. F$id returns the owner number (user Index) of the current process (that's us). F$suser changes the user number to whatever you say (no questions asked.)

USERID is callable by BASIC09* and accepts one or two integer values. The first (and maybe only) variable returns the current user index. If there is a second integer variable passed to it, this value is used to set the userid of the process. The main PASSWORD program passes a zero in this second value. This makes the process match the owner of the password file and allows it to read, copy and delete it.

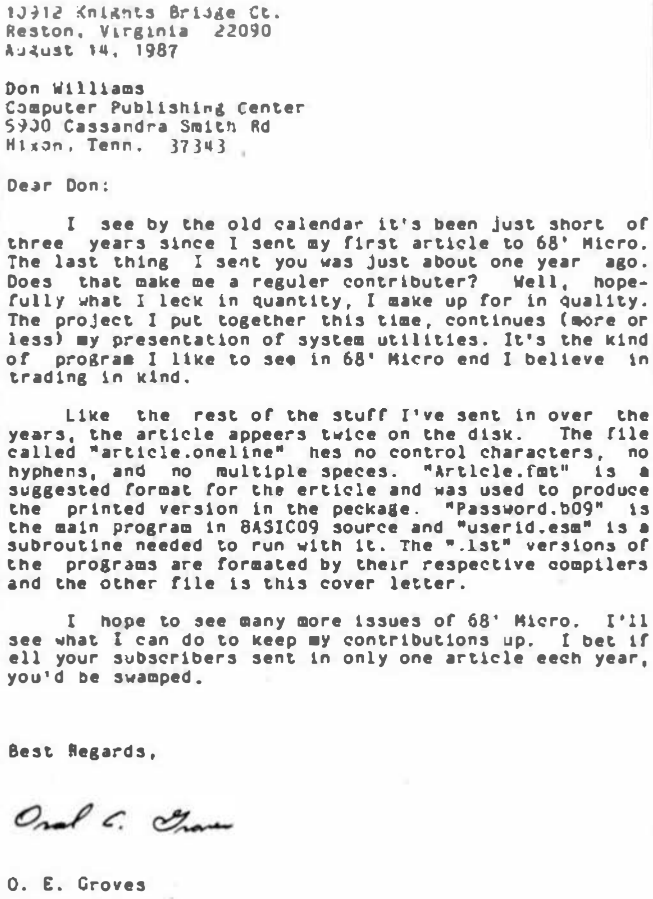
I tried to make PASSWORD as general as possible, but there are some things to watch out for. The two file variables, TFILE and PFILE would have to point to your system device. Password uses shell commands, TMODE, ATTR and RENAME. These must be available in either the module or execution directories. I could be accused of taking the tools concept a little too far at this point. Yes, what I wanted could have been done from inside BASIC09*, but why make a program that is only used once in a while more complex than it has to be? The only shell command I hesitated to use was ATTR. There ought to be a way to tell BASIC09* not to make a file public read. I looked through the documentation, but couldn't turn up anything. The last thing that needs mentioning is, by setting everything to upper case I don't have to mess with mixed mode characters. It also means that ALL USER DATA in the password file (ie. name and password) must be UPPERCASE. Also the password file is left around as long as possible. This means that up to the end there is at least one copy of it on the system.

There are some things that working on this program and especially on this article have brought to the surface. Security is a sword with two edges. As the system manager you are obligated to keep your users' files out of harm's way. This includes not letting anyone but the owner mess with them. In trying to do that though, you wind up in a position do that very same kind of messing yourself. I could have made the program much simpler if I had just assumed that the person running it would be authorized to use it. Leaving the password file with public access is the most straight forward way to do that. This is unacceptable on the grounds that if anyone's password can be read, it gives access to their files. However, by trying to get better control I have opened a whole new can of worms. The implication of the USERID routine is that I have now made it easier to get to those very files I have been trying to protect. The routine can be used in any BASIC09* program to not only access a file but change it so that the rightful owner can't even get to it! I had to be very careful in the design of password to make sure that a user was really who he says he is.

In my work with PASSWORD, I have explored the tools concept. This came in the form of shell commands run from inside a BASCI09 program. I pursued the use of a BASIC09* callable assembly language subroutine. Coincidentally, the program became a point from which to take a look at small system security. Like most programming projects, this one has been both interesting and an education. Questions and comments are welcome, and may be addressed to:

O. E. Groves
10912 Knights Bridge Ct.
Reston, Virginia 22090

* OS-9 is a registered trademark of Microware Systems Incorporated. BASCI09 is a trade mark of Microware and Motorola Inc. COCO refers to the Color Computer, a Product of Tandy Radio Shack.



10912 Knights Bridge Ct.
Reston, Virginia 22090
August 14, 1987

Don Williams
Computer Publishing Center
5900 Cassandra Smith Rd
Hixon, Tenn. 37343

Dear Don:

I see by the old calendar it's been just short of three years since I sent my first article to 68' Micro. The last thing I sent you was just about one year ago. Does that make me a reguler contributer? Well, hopefully what I leck in quantity, I make up for in quality. The project I put together this time, continues (more or less) my presentation of system utilities. It's the kind of program I like to see in 68' Micro end I believe in trading in kind.

Like the rest of the stuff I've sent in over the years, the article appeers twice on the disk. The file called "article.oneline" hes no control characters, no hyphens, and no multiple speces. "Article.fmt" is a suggested format for the erticle and was used to produce the printed version in the peckage. "Password.b09" is the main program in BASIC09 source and "userid.esm" is a subroutine needed to run with it. The ".lst" versions of the programs are formated by their respective compilers and the other file is this cover letter.

I hope to see many more issues of 68' Micro. I'll see what I can do to keep my contributions up. I bet if ell your subscribers sent in only one article eech year, you'd be swamped.

Best Regards,

Oral E. Groves

O. E. Groves

Editor's Note: Thanks Oral for the nice words. I wholeheartedly agree. I guess some of the new comers don't understand where we came from. I wish everyone would keep in mind that we are a "contributor supported" magazine. Without the support and input of folks such as yourself, we would have closed the doors a long time ago! It is a paradoxical condition, but at this stage in the game we are the third oldest computer magazine in existence, the "Granddaddy" of the present rage "desktop publishing" and have more readers and distribution and less advertisers than at any other time in our 10 year plus history.

I also hope to see many more issues of 68 Micro Journal, but I must admit, we not only need article input, but could sure stand some new advertising blood. Even in our time of need we have just recently refused advertising because the product didn't meet the standards we set years ago. I still believe, despite lawsuits, poor or no profits, dwindling advertisers and all the other adversities we are presently going through, because of loyal folks such as yourself, we will be around for quite some time to come!

Thanks Oral - God Bless!

DMW

```
PROCEDURE PASSWORD

(* This program changes the current user's password in the /v0/sys/
(* password file.  If the caller is user zero ,the password for any
(* other user may be changed.
(*
(* AUTHOR        DATE VERSION              COMMENT
(*
(* OE GROVES   07/10/87  1    INITIAL VERSION
(* OE GROVES   07/25/87  2    File names converted to variables.
(*                            Fix userid routine
(*
(*   Released into public domain 1987 - OE Groves
(*

(* define and initialze variables

DIM unam,pswd:STRING[32]; uid,pri:INTEGER; XDIR,DDIR,EX:STRING[32]
DIM NUSER:INTEGER
DIM OUSER,OPATH,IPATH,PROCADD,I,N:INTEGER
DIM OPASS,NPASS,TPASS,NAME,PFILE,TFILE:STRING[32]
DIM OKAY:BOOLEAN
ON ERROR GOTO 10

(*   These file name will need to be changed to fit another system

PFILE = "/V0/SYS/PASSWORD"
TFILE = "/VO/SYS/PASSTEMP"

IPATH = 0
OPATH = 0
ouser=0
nuser=0
OKAY = FALSE
I=0
opass = ""
npass = "$"

(* get user number from process descriptor and change to user 0
run userid(ouser,nuser)

(* have all input converted to uppercase
shell "tmode .1 upc"

(* get user information from the terminal
WHILE (I<4) AND (TPASS <> NPASS) DO
(*   turn on echo of user name
 shell "tmode .1 echo"
 INPUT "WHAT IS YOUR USER ID NAME? ",NAME

(*   turn off echo of password data
 shell "tmode .1 -echo"

 INPUT "WHAT Is YOUR OLD PASSWORD? ",OPASS
 INPUT "ENTER NEW PASSWORD PLEASE. ",NPASS
 INPUT "ENTER IT ONCE MORE PLEASE. ",TPASS
 IF TPASS <> NPASS THEN
  PRINT
  PRINT "THE SECOND TIME WASN'T THE SAME"
 ENDIF
 IF OPASS = NPASS THEN
  PRINT
  PRINT "CAN'T USE THE OLD PASSWORD AGAIN"
  TPASS = NPASS+"*"
 ENDIF
```

```
I=I+1
ENDWHILE
IF I => 4 THEN
  PRINT
  PRINT "CHECK YOUR INFORMATION AND START AGAIN"
  ERROR $6
ENDIF
(*  Copy the password file while looking for match to
(*    user name and password
CREATE #Opath,TFILE:UPDATE
OPEN #Ipath,PFILE:READ
WHILE NOT(EOF(#Ipath)) DO
 INPUT #Ipath,unam,pswd,uid,pri,XDIR,DDIR,EX
 IF (UNAM = NAME) AND (PSWD = OPASS) THEN
  IF (OUSER = 0) OR (UID = OUSER) THEN
    PSWD = NPASS
    OKAY = TRUE
  ENDIF
 ENDIF
 PRINT #Opath,unam;",";pswd;",";uid;",";pri;",";XDIR;",";DDIR;",";EX
ENDWHILE
CLOSE #IPATH
ipath = 0
CLOSE #OPATH
opath = 0
DELETE PFILE
SHELL "attr " + TFILE + " -pr;RENAME " + TFILE + " password"

(*   if no match found, print message
IF NOT(OKAY) THEN
 PRINT "THAT IS NOT YOUR USER ID. PASSWORD NOT CHANGED"
ENDIF
(* fix terminal output
shell "tmode .1 -upc echo"
END

(* handle errors
10 N=ERR
IF N<>0 THEN
  PRINT
  PRINT "ERROR ---> ";N
  PRINT "NO CHARGES MADE"
  PRINT
ENDIF
IF IPATH <> 0 THEN
 CLOSE #IPATH
ENDIF
IF OPATH <> 0 THEN
 CLOSE #OPATH
 DELETE TFILE
ENDIF
shell "tmode .1 -upc echo"
```

```
          B:PROCEDURE PASSWORD
0000
0001 (* This program changes the current user's password in the /v0/sys/
0044 (* password file.  If the caller is user zero ,the password for any
0087 (* other user may be changed.
00A4 (*
00A7 (* AUTHOR        DATE VERSION              COMMENT
00D8 (*
000B (* OE GROVES   07/10/87  1    INITIAL VERSION
0108 (* OE GROVES   07/25/87  2    File names converted to variables.
0148 (* Fix userid routine
015D (*
0160 (* Released into public domain 1987 - OE Groves
018F (*
0192
0193 (* define and initialze variables
01B4
01B5 DIM unam,pswd:STRING[32]; uid,pri:INTEGER; XDIR,DDIR,EX:STRING[32]
01E2 DIM NUSER:INTEGER
01E9 DIM OUSER,OPATH,IPATH,PROCADD,I,N:INTEGER
0204 DIM OPASS,NPASS,TPASS,NAME,PFILE,TFILE:STRING[32]
0224 DIM OKAY:BOOLEAN
022B ON ERROR GOTO 10
0231
0232 (* These file name will need to be changed to fit another system
0272
0273 PFILE="/V0/SYS/PASSWORD"
028A TFILE="/V0/SYS/PASSTEMP"
02A1
02A2 IPATH=0
02A9 OPATH=0
02B0 OUSER=0
02B7 NUSER=0
02BE OKAY=FALSE
02C4 I=0
02CB OPASS="*"
02D3 NPASS="&"
02DB
02DC (* get user number from process descriptor and change to user 0
031B RUN userid(OUSER,NUSER)
032A
032B (* have all input converted to uppercase
0353 SHELL "tmode .1 upc"
0363
0364 (* get user information from the terminal
038D WHILE I<4 AND TPASS<>NPASS DO
03A1   (* turn on echo of user name
03BD   SHELL "tmode .1 echo"
03CE   INPUT "WHAT IS YOUR USER ID NAME? ",NAME
03F1
03F2   (* turn off echo of password data
0413   SHELL "tmode .1 -echo"
0425
0426   INPUT "WHAT IS YOUR OLD PASSWORD? ",OPASS
0449   INPUT "ENTER NEW PASSWORD PLEASE. ",NPASS
046C   INPUT "ENTER IT ONCE MORE PLEASE. ",TPASS
048F   IF TPASS<>NPASS THEN
049C     PRINT
049E     PRINT "THE SECOND TIME WASN'T THE SAME"
04C1   ENDIF
04C3   IF OPASS=NPASS THEN
04D0     PRINT
04D2     PRINT "CAN'T USE THE OLD PASSWORD AGAIN"
04F6     TPASS=NPASS+"*"
0502   ENDIF
0504   I=I+1
050F ENDWHILE
0513 IF I>=4 THEN
051F   PRINT
0521   PRINT "CHECK YOUR INFORMATION AND START AGAIN"
054B   ERROR 56
054F ENDIF
0551 (* Copy the password file while looking for match to
0586 (* user name and password
```

```
059F CREATE #OPATH,TFILE:UPDATE
05AB OPEN #IPATH,PFILE:READ
05B7 WHILE NOT(EOF(#IPATH)) DO
05C2   INPUT #IPATH,unam,pswd,uid,pri,XDIR,DDIR,EX
05E4   IF unam=NAME AND pswd=OPASS THEN
05F0     IF [illegible]=0 OR uid=OUSER THEN
060D  pswd=NPASS
0615  OKAY=TRUE
061B     ENDIF
061D   ENDIF
061F   PRINT #OPATH,unam; ",": pswd; ",": uid; ",": pri; ",": XDIR; ",": DDIR; ",": EX
0659 ENDWHILE
065D CLOSE #IPATH
0663 IPATH=0
066A CLOSE #OPATH
0670 OPATH=0
0677 DELETE PFILE
067C SHELL "attr "+TFILE+" -pr;RENAME "+TFILE+" password"
06A8
06A9 (* if no match found, print message
06CC IF NOT(OKAY) THEN
06D6   PRINT "THAT IS NOT YOUR USER ID. PASSWORD NOT CHANGED"
0708 ENDIF
070A (* fix terminal output
0720 SHELL "tmode .1 -upc echo"
0736 END
0738
0739 (* handle errors
0749 10    N=ERR
0752       IF N<>0 THEN
075E         PRINT
0760         PRINT "ERROR ---> "; N
0773         PRINT "NO CHANGES MADE"
0786         PRINT
0788       ENDIF
078A       IF IPATH<>0 THEN
0796         CLOSE #IPATH
079C       ENDIF
079E       IF OPATH<>0 THEN
07AA         CLOSE #OPATH
07B0         DELETE TFILE
07B5       ENDIF
07B7       SHELL "tmode .1 -upc echo"

*
*
*    PROCEDURE USERID
*
* BASIC09 callable subroutine to get/change caller's USER ID.
*
*    Syntax:
*    RUN userid(oldid,newid)  Where oldid and newid are integer
*        variables and newid is optional if change is wanted.
*
*
f$suser set $1c define system call
vers set 1 version number
type set sbrtn+objct
rev set reent+1
*
 mod uidend,uidname,type,rev,uidstrt,0
 ifp1
use /d1/elektra_defs/os9defs
endc
*
```

```
uidname fcs /userid/ module name
 fcb vers
*
*    define stack areas
*
 org 0
return rmb 2 subroutine return addrs
count rmb 2 # of parms passed from BASIC09
olduid rmb 2 addr of 1st parm
oldsiz rmb 2 byte count of 1st parm
newuid rmb 2 addr of 2nd parm
newsiz rmb 2 byte count of 2nd parm
*
uidstrt ldd count,s any parms?
 beq uiderr problem
 ldd oldsiz,s will user num fit?
 cmpd #2 must be integer
 bne uiderr problem
 os9 f$id sys call for procid and userid
 sty [olduid,s] put userid away
 ldd count,s how many parms?
 decb
 beq uidrtn need only return old uid
 ldd newsiz,s
 cmpd #2 must be integer
 bne uiderr
 ldy [newuid,s] get new user number
 os9 f$suser set new user id
uidrtn rts
*
* parameter error return
*
uiderr comb set carry
 ldb #56 parameter err number
 rts
 emod
uidend equ *
 end

Microware OS-9 Assembler 2.1   08/16/87 12:46:53                Page 001
 - OS-9 System Symbolic Definitions

00001          *
00002          *
00003          *    PROCEDURE USERID
00004          *
00005          * BASIC09 callable subroutine to get/change caller's USER ID.
00006          *
00007          *    Syntax:
00008          *    RUN userid(oldid,newid)  Where oldid and newid are integer
00009          *        variables and newid is optional if change is wanted.
00010          *
00011          *
00012   001C             f$suser  set   $1c        define system call
00013   0001             vers     set   1          version number
00014   0021             type     set   sbrtn+objct
00015   0081             rev      set   reent+1
00016          *
00017   0000 87CD0043             mod   uidand,uidname,type,rev,uidstrt,0
00018                             ifp1
                                        /d0/defs/os9defs
00020                             endc
00021          *
```

```
00022    000D 75736572   uidname  fcs   /userid/   module name
00023    0013 01                  fcb   vers
00024         *
00025         *    define stack areas
00026         *
00027 D 0000                      org   0
00028 D 0000             return   rmb   2          subroutine return addrs
00029 D 0002             count    rmb   2          # of parms passed from BASIC09
00030 D 0004             olduid   rmb   2          addr of 1st parm
00031 D 0006             oldsiz   rmb   2          byte count of 1st parm
00032 D 0008             newuid   rmb   2          addr of 2nd parm
00033 D 000A             newsiz   rmb   2          byte count of 2nd parm
00034         *
00035    0014 EC62       uidstrt  ldd   count,s    any parms?
00036    0016 2724                beq   uiderr     problem
00037    0018 EC66                ldd   oldsiz,s   will user num fit?
00038    001A 10830002            cmpd  #2         must be integer
00039    001E 261C                bne   uiderr     problem
00040    0020 103F0C              os9   f$id       sys call for procid and userid
00041    0023 10AFF804            sty   [olduid,s] put userid away
00042    0027 EC62                ldd   count,s    how many parms?
00043    0029 5A                  decb
00044    002A 270F                beq   uidrtn     need only return old uid
00045    002C EC6A                ldd   newsiz,s
00046    002E 10830002            cmpd  #2         must be integer
00047    0032 26D8                bne   uiderr
00048    0034 10AEF808            ldy   [newuid,s] get new user number
00049    0038 103F1C              os9   f$suser    set new user id
00050    003B 39         uidrtn   rts
00051         *
00052         * parameter error return
00053         *
00054    003C 53         uiderr   comb             set carry
00055    003D C638                ldb   #56        parameter err number
00056    003F 39                  rts
00057    0040 7B2D59              emod
00058    0043            uidend   equ   *
00059                             end

00000 error(s)
00000 warning(s)
$0043 00067 program bytes generated
$000C 00012 data bytes allocated
```

EOF

The "Grandfather of Desk Top Publishing"

Ramblings
&
What-not...

The Editor's Place to Comment!

Have you ever read something and gotten the feeling that you wanted to get up and spit? Well, I did while catching up on my reading a little earlier this evening. Actually yesterday evening as it is now into the wee hours of the next day.

First, I ran across a Tandy ad that caught my eye. Why? Because it repeats a myth I have seen them recite in print previously.

> "In 1977, we became the first company to *successfully* (italics ours) manufacture and market a personal computer - nobody's been in the business longer." (Tandy)

As thousands of us know - *Bull-Hockey! Not so!*

While Tandy and their Radio Shack stores were floggin' off CB's, stereos and other non-digital stuff, there were several domestic companies manufacturing personal computers (?)-whatever personal computer (PC) means — and successfully advertising and marketing them world-wide! Most all of us remember a few like MITS, SWTPC (still in there beating the bushes), Sphere, etc. Actually there were other brands that were manufactured and sold while the guys who wrote those dumb advertising copy lines were still "tinkling" in their pin ons! However, to sorta look halfway honest they slip in a *more or less* word like "successfully". So to be "real" I guess you have to achieve a certain degree or level of "*successfully*". Little guys don't count.

As I see it, some are starting to puff and spout things like, "we were first" or "we started it all", kinda ballyhoo. However, if you're going to shoot from the hip you certainly should have some, however slight, basis in fact, or, as my Western friends would say-"dry powder and hard lead". Without that you're just dumping *vapor-fact(s)*", *take note, another new word* I just invented. You can tell everyone you know you saw it here first! And maybe last...

No denying Tandy has made pretty big waves in the micro world. They have flogged off some pretty neat things, computers and what-not. But they don't need to try to "bull" us. We appreciate them for what they are, not what someone else is, or was.

About the PC word part I don't know. PC was being used long before Tandy ever sold their first micro-computer. However, as to being the first, well, that's a pretty big bunch of vapor-fact(s) to be dishing out to some of us who can remember when—and who! A lot of folks have been in the business considerably longer!

After re-reading the above comments I am beginning to like my new word "vapor-fact(s)". Can't seem to let it go. So guess I'll let you in on another prime example of vapor-fact(s).

Seems Apple, not to be outdone, also likes to trot out a vapor-fact(s) every now and then. Example, a recent Apple two page advertising spread said, in part:

> "despite the acclaim we received for having *created* (italics ours) desktop publishing..." (Apple)

Now how 'bout them beans? Apple *created* desktop publishing like Columbus discovered hot chocolate, M&M's and Hoola Hoops! However, it is a fact, Apple sure did more than about anyone else in the *improvement* department! And that is the truth of it-Apple has done more than all others combined to make it a better process, but create it...never!

Now as most of you know, we published in 68 Micro Journal, back in the late 70's, that we did and would continue to publish 68 Micro Journal only with the same type computers you could buy as a personal computer (?), and those we published articles about. Nothing has changed since then.

That being the facts. "68 Micro Journal was the first product of what is known now as *desktop publishing*." That being ***not*** a "vapor-fact(s)".

But wait. "what is desktop publishing?". I ask? Nobody here in these wee hours except Charlie, Buster and myself. Charlie being the typical poodle pays little attention to such mundane topics of conversation and Buster, a young but lazy tom cat, sleeps all the time and wouldn't hear a PC hit the floor if dropped from 20 feet, and crashed a foot from his head. So I guess I will have to give you the *real* answer, rather than have you ponder it to some unlikely conclusion.

"Desktop publishing" is publishing done on a desktop!
Right? Right!
Well, sorta.

Not to be confused with publishing done on the floor (large, expensive hard to program machines that take up the better part of an average room, or more, and sit on the floor of course) We got one of 'em also.That, I guess, is floor-top publishing. Or maybe just floor publishing. Doesn't sound too glamourous so I guess thats why they never called it anything but just plain old "publishing." And nobody seems to really care about that claim of being first, unless we go back several hundred years. Also one wouldn't or at least shouldn't confuse desktop publishing with "deskdrawer publishing", a word heard every now and then. That, I imagine, is where you would publish with something that slides into a desk cubicle or drawer. However, that doesn't seem to qualify either. Hard to get man (or lady) and machine in there.

We could really dig into this thing, for a long time to come, with all sorts of claims like the first to do "no pencil publishing", or how about "bathtub publishing?" Bet no one has thought of that yet. Or "one person publishing" would have been much better. And as we get more into digital publishing, that is, direct from input to the press plates (actually coming soon), then someone will probably call it "digital publishing." Pretty simple, huh? Once you understand how to figure it.

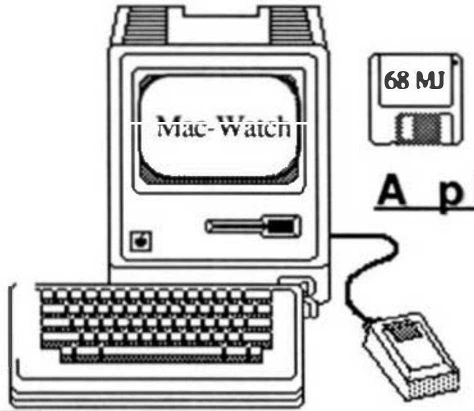
When we sift through the whole mess it still boils down to a pretty simple solution, **"desktop publishing" is done on an *imaginary* desktop! Or to be more precise, in an area much, much smaller than traditional "floor" publishing.** I know that's too simplified, however, it was an expression to denote the entire process could be done in a space or area so small as to be unheard of at the time it came into being. Even so compacted as to get everything on the average desktop. Now, that was something to crow about. I ought to know, I was publishing a newspaper with traditional floor publishing. And I was determined that this new technology would fulfill it's promise. At least for me. For that reason, and of course, the economics of everything in a small size (comparative), one man (lady) easily manipulated system. I started 68 Micro Journal, in part, to show it could be done. That meant designing and building the necessary hardware and writing the original software, because none was commercially available at that time-**NONE!** While the technology has advanced in quantum leaps, in comparison, what we had when I started and what we have available now, pales in the comparison of where the first "Wright Flyer" was technically and the present day space shuttle technology. But, still the space shuttle has as it's roots, the Wright Flyer. As concerns what we now call "desktop publishing", we didn't give it it's name, *we just gave it birth.* Others occasionally try to make that into something else. Oh, well.

All along the title or name actually should have been— "electronic publishing". That is actually what the difference is, electronic as opposed to non-electronic or partial electronic. With a simple name like that nobody would want to tarnish their reputation by risking getting caught trying to lay claim to someones else's efforts. Although I suspect some of those other good *old boys would still maybe give it a try.*

So we, 68 Micro Journal, by virture of truth and proof, are acknowledged to be the first "real" product of "desktop publishing", and "The fountainhead from which it sprang". "Granddaddy" or "Father" of desktop publishing, to the less romantic of you. That and a couple of bucks will get you a cup of coffee in a lot of places.

Now, let's just sit around and see who tries to rip off that "fountain-head" stuff. Remember, if all this seems like garbage, well, did you ever hear of "vapor-fact(s)" before?

Gotcha there...!



The Macintosh Section™

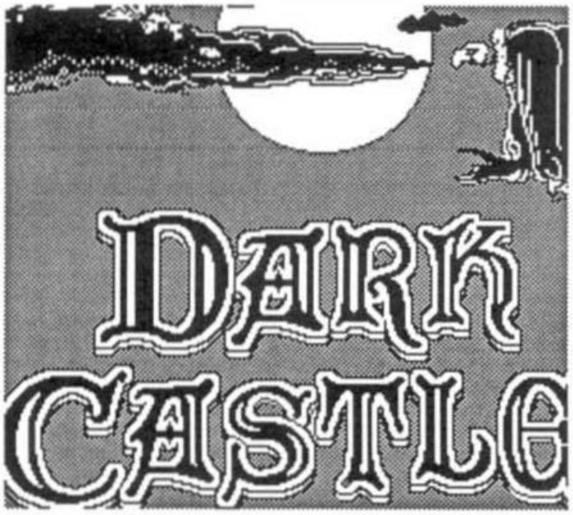
Reserved as a

A place for your thoughts

And Ours......

Mac-Watch

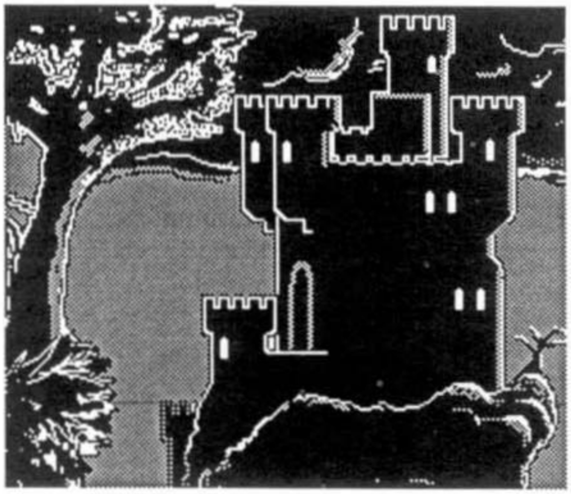
Lightning strikes, thunder echoes, haunting music plays, the drawbridge creaks open, and you enter the fast-paced, high adventure world of Dark Castle, the arcade-style game from Silicon Beach Software.



With graphics better than most arcade games, you run, jump, climb, and swing your way through fourteen separate rooms, each of which offers different challenges: gargoyles, mutants, dragons, guards, bats, wizards, and fireballs, to name only a few. The graphics are so intricate and smooth-flowing that the prisoners being whipped by the dungeon's torturer turn their heads to watch as you pick up a mace and duel with the henchman.

The object of Dark Castle is to make your way through the four castle areas gathering rocks, elixir, shields, and fireballs with the ultimate goal of toppling the Block Knight. Unlike text adventures which require you to begin at scene one and move progressively, Dark Castle allows you to choose among four doors, each with a wide variety of adventures.

This is probably the best graphics & sound of any Macintosh program now available it has to be seen to be believed! DMW



From the Great Hall where each round begins, Door One leads to "Trouble" rooms 1, 2, and 3. You go into the depths

Telephone: (615) 842-4600 **South East Media** Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

SOFTWARE

!!! Please Specify Your Operating System and Disk Size !!!

SCULPTOR

Full OEM & Dealer Discounts Available!

THE SCULPTOR SYSTEM

Sculptor combines a powerful fourth generation language with an efficient database management system. Programmers currently using traditional languages such as Basic and Cobol will be amazed at what Sculptor does to their productivity. With Sculptor you'll find that what used to take a week can be achieved in just a few hours.

AN ESTABLISHED LEADER

Sculptor was developed by professionals who needed a software development tool with capabilities that were not available in the software market. It was launched in 1981 and since then, with feedback from an ever-increasing customer base, Sculptor has been refined and enhanced to become one of the most adaptable, fast, and above all reliable systems on the market today.

SYSTEM INDEPENDENCE

Sculptor is available on many different machines and for most operating systems, including MS DOS, Unix/Xenix and VMS. The extensive list of supported hardware ranges from small personal computers, through multi-user micros up to large minis and mainframes. Sculptor is constantly being ported to new systems.

APPLICATION PORTABILITY

Mobility of software between different environments is one of Sculptor's major advantages. You can develop applications on a stand-alone PC and – without any alterations to the programs – run them on a large multi-user system. For software writers this means that their products can reach a wider marketplace than ever before. It is this system portability, together with high-speed development, that makes Sculptor so appealing to value added resellers, hardware manufacturers and software developers of all kinds.

SPEED AND EFFICIENCY

Sculptor uses a fast and proven indexing technique which provides instant retrieval of data from even the largest of files. Sculptor's fourth generation language is compiled to a compact intermediate code which executes with impressive speed.

INTERNATIONALLY ACCEPTED

By using a simple configuration utility, Sculptor can present information in the language and format that you require. This makes it an ideal product for software development almost anywhere in the world. Australasia, the Americas and Europe – Sculptor is already at work in over 20 countries.

THE PACKAGE

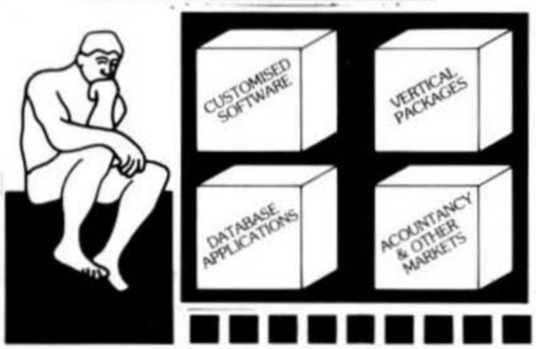
With every development system you receive:

□ A manual that makes sense
□ A periodic newsletter
□ Screen form language
□ Report generator
□ Menu system
□ Query facility
□ Set of utility programs
□ Sample programs

For resale products, the run-time system is available at a nominal cost.

Facts **Features**

Sculptor for 68020
OS-9 & UniFLEX
$995



DATA DICTIONARY

Each file may have one or more record types described. Fields may have a name, heading, type, size, format and validation list. Field type may be chosen from:

□ alphanumeric
□ integer
□ floating point
□ money
□ date

DATA FILE STRUCTURE

□ Packed, fixed-length records
□ Money stored in lower currency unit
□ Dates stored as integer day numbers

INDEXING TECHNIQUE

Sculptor maintains a B-tree index for each data file. Program logic allows any numbers of alternative indexes to be coded into one other file.

INPUT DATA VALIDATION

Input data may be validated at three levels:

□ automatic by field type
□ validation list in data dictionary
□ programmer coded logic

ARITHMETIC OPERATORS

–	Unary minus
*	Multiplication
/	Division
%	Remainder
+	Addition
–	Subtraction

RELATIONAL OPERATORS

=	Equal to
<	Less than
>	Greater than
< =	Less than or equal to
> =	Greater than or equal to
< >	Not equal to
and	Logical and
or	Logical or
ct	Contains
bw	Begins with

SPECIAL FEATURES

□ Full date arithmetic
□ Echo suppression for passwords
□ Terminal and printer independence
□ Parameter passing to sub-programs
□ User definable date format

MAXIMA AND MINIMA

Minimum key length	1 byte
Maximum key length	160 bytes
Minimum record length	3 bytes
Maximum record length	32767 bytes
Maximum fields per record	32767
Maximum records per file	16 million
Maximum files per program	16
Maximum open files	Operating system limit

PROGRAMS

□ Define record layout
□ Create new indexed file
□ Generate standard screen-form program
□ Generate standard report program
□ Compile screen-form program
□ Compile report program
□ Screen-form program interpreter
□ Report program interpreter
□ Menu interpreter

□ Query facility
□ Reformat file
□ Check file integrity
□ Rebuild index
□ Alter language and date format
□ Setup terminal characteristics
□ Setup printer characteristics

SCREEN-FORM LANGUAGE

□ Programmer defined options and logic
□ Multiple files open in one program
□ Default or programmer processing of exception conditions
□ Powerful verbs for input, display and file access
□ Simultaneous display of multiple records
□ Facility to call sub-programs and operating system commands
□ Conditional statements
□ Subroutines
□ Independent of terminal type

MUSTANG-020 Users - Ask For Your Special Discount!

* Tandy CoCo III Special - Reg. $595 * Special $389 *

	*	**	***		*	**	***
MUSTANG-020	$995	$199	$595	PC/XT/AT MSDOS	$595	$119	$595
OS/9 UniFLEX 6809	"	"	"	AT&T 3B1 UNIX	"	"	"
IBM Compatibles	"	"	"	SWTPC 68010 UniF	$1595	$319	$797
Tandy CoCo III	Special $389.00			SWTPC 68010 UNIX	$1990	$398	$995

... Sculptor Will Run On Over 100 Other Types of Machines ...

... Call for Pricing ...

!!! Please Specify Your Make of Computer and Operating System !!!

* Full Developement Package
** Run Time Only
*** C Key File Library

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343
Telephone: (615) 842-4600 Telex: 5106006630

master card

** Shipping **
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600

South East Media

Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

DISASSEMBLERS

SUPER SLEUTH from Computer Systems Consultants Interactive Disassembler; extremely *POWERFUL!* Disk File Binar /ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems.

Color Computer SS-50 Bus (all w/ A.L. Source)
CCD (32K Req'd) Obj. Only $49.00
F, S, $99.00 - CCF, Obj. Only $50.00 U, $100.00
CCF, w/Source $99.00 O, $101.00
CCO, Obj. Only $50.00
OS9 68K Obj. $100.00 w/Source $200.00

DYNAMITE+ – Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" Files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only $100.00 - CCO, Obj. $ 59.95
F, S, " " $100.00 - O, object only $150.00
U, " " $300.00

PROGRAMMING LANGUAGES

PL/9 from Windrush Micro Systems -- By Graham Trott. A combination Editor Compiler Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit Integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. 500+ page Manual with tutorial guide.

F, S, CCF - $198.00

PASC from S.B. Media - A FLEX9, SK*DOS Compiler with a definite Pascal "flavor". Anyone with a bit of Pascal experience should be able to begin using PASC to good effect in short order. The PASC package comes complete with three sample programs: ED (a syntax or structure editor), EDITOR (a simple, public domain, screen editor) and CHESS (a simple chess program). The PASC package come complete with source (written in PASC) and documentation.

*FLEX, SK*DOS $95.00*

WHIMSICAL from S.E. MEDIA Now supports *Real Numbers.* "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit Integers, 6-digit Real Numbers, unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer, etc. Run-Time subroutines inserted as called during compilation. *Normally produces 10% less code than PL/9.*

F, S and CCF - $195.00

KANSAS CITY BASIC from S.E. Media - *Basic for Color Computer OS-9* with many new commands and sub-functions added. A full implementation of the IF-THEN-ELSE logic is included, allowing nesting to 255 levels. Strings are supported and a subset of the usual string functions such as LEFT$, RIGHT$, MID$, STRING$, etc. are included. Variables are dynamically allocated. Also included are additional features such as Peek and Poke. A must for any Color Computer user running OS-9.

CoCo OS-9 $39.95

C Compiler from Windrush Micro Systems b James McCosh. Full C for FLEX, SK*DOS except bit-fields, including an Assembler. *Requires the TSC Relocating Assembler if user desires to implement his own Libraries.*

F, S and CCF - $295.00

C Compiler from Introl – Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler, FAST, efficient Code. More UNIX Compatible than most.

*FLEX, SK*DOS, CCF, OS-9 (Level II ONLY), U - $575.00*

PASCAL Compiler from Lucidata – ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.

F, S and CCF 5" - $190.00 F, S 8" - $205.00

PASCAL Compiler from OmegaSoft (now Certified Software) -- For the *PROFESSIONAL*; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible. Requires a "Motorola Compatible" Relo. Asmb. and Linking Loader.

F, S and CCF - $425.00 - One Year Maint. $100.00
OS-9 68000 Version - $900.00

KBASIC - from S.E. MEDIA -- A "Native Code" BASIC Compiler which is now Fully TSC XBASIC compatible. The compiler compiles to Assembly Language Source Code. A NEW, streamlined, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.

*FLEX, SK*DOS, CCF, OS-9 Compiler /Assembler $99.00*

CRUNCH COBOL from S.E. MEDIA -- Supports large subset of ANSII Level 1 *COBOL* with many of the useful Level 2 features. Full FLEX, SK*DOS File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System. *A very popular product.*

*FLEX, SK*DOS, CCF - $99.95*

FORTH from Stearns Electronics -- A CoCo FORTH Programming Language. Tailored to the CoCo! Supplied on Tape, transferable to disk. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!

Color Computer ONLY - $58.95

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

** Shipping **
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600

South East Media

Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

FORTHBUILDER is a stand-alone target compiler (crosscompiler) for producing custom Forth* systems and application programs. All of the 83-standard defining words and control structures are recognized by FORTHBUILDER.

FORTHBUILDER is designed to behave as much as possible like a resident Forth interpreter/compiler, so that most of the established techniques for writing Forth code can be used without change.

Like compilers for other languages, FORTHBUILDER can operate in "batch mode".

The compiler recognizes and emulates target names defined by CONSTANT or VARIABLE and is readily extended with "compile-time" definitions to emulate specific target words.

FORTHBUILDER is supplied as an executable command file configured for a specific host system and target processor. Object code produced from the accompanying model source code is royalty-free to licensed users.

F, CCF, S - $99.95

DATABASE ACCOUNTING

XDMS from Westchester Applied Business Systems

FOR 6809 FLEX-SK*DOS(5/8")

Up to 32 groups/fields per record! Up to 12 character filed name! Up to 1024 byte records! User defined screen and print control! Process files! Form files! Conditional execution! Process chaining! Upward/ Downward file linking! File joining! Random file virtual paging! Built in utilities! Built in text line editor! Fully session oriented! Enhanced forms! Boldface, Double width, Italics and Underline supported! Written in compact structured assembler! Integrated for FAST execution!

XDMS-IV Data Management System

XDMS-IV is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, but also to process entire files producing customized reports, screen displays and file output. Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" on user defined output reports.

POWERFUL COMMANDS!

XDMS-IV combines the functionality of many popular DBMS software systems with a new easy to use command set into a single integrated package. We've included many new features and commands including a set of general file utilities. The processing commands are Input-Process-Output (IPO) orierve which allows almost instant implementation of a process design.

SESSION ORIENTED!

XDMS-IV is session oriented. Enter "XDMS" and you are in instant command of all the features. No more waiting for a command to load in from disk! Many commands are immediate, such as CREATE (file definition), UPDATE (file editor), PURGE and DELETE (utilities). Others are process commands which are used to create a user process which is executed with a RUN command. Either may be entered into a "process" file which is executed by an EXECUTE statement. Processes may execute other processes, or themselves, either conditionally or unconditionally. Menus and screen prompts are easily coded, and entire user applications can be run without ever leaving XDMS-IV

IT'S EASY TO USE!

XDMS-IV keeps data management simple! Rather than design a complex DBMS which hides the true nature of the data, we kept XDMS-IV file oriented. The user view of data relationships is presented in reports and screen output, while the actual data resides in easy to maintain files. This aspect permits customized presentation and reports without complex redefinition of the database files and structure. XDMS-IV may be used for a wide range of applications from simple record management systems (addresses, inventory ...) to integrated database systems (order entry, accounting...)

The possibilities are unlimited...

FOR 6809 FLEX-SK*DOS(5/8") **$249.95**

ASSEMBLERS

ASTRUK09 from S.E. Media -- A "Structured Assembler for the 6809" which requires the TSC Macro Assembler.

F, S, CCF - $99.95

Macro Assembler for TSC -- The FLEX, SK*DOS STANDARD Assembler.

Special -- CCF $35.00; F, S $50.00

OSM Extended 6809 Macro Assembler from Lloyd I/O. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. GeneÒrate OS-9 Memory modules under FLEX, SK*DOS.

*FLEX, SK*DOS, CCF, OS-9 $99.00*

Relocating Assembler/Linking Loader from TSC. -- Use with many of the C and Pascal Compilers.

F, S, CCF $150.00

MACE, by Graham Trott from Windrush Micro Systems -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs.

F, S, CCF - $75.00

XMACE -- MACE w/Cross Assembler for 6800/1/2/3/8

F, S, CCF - $98.00

Telephone: (615) 842-4600

South East Media

Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

UTILITIES

Basic09 XRef from S.E. Media -- This Basic09 Cross Reference Utility is a Basic09 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a Program List Utility which outputs a fast "pretty printed" listing with line numbers. Requires Basic09 or RunB.

O & CCO obj. only -- $39.95; w/ Source - $79.95

BTree Routines - Complete set of routines to allow simple implementation of keyed files - *for your programs* - running under Basic09. A real time saver and should be a part of every serious programmers tool-box.

O & CCO obj. only - $89.95

Lucidata PASCAL UTILITIES (Requires Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

INCLUDE -- Include other Files in a Source Text, including Binary - unlimited nesting.

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.

F, S, CCF --- EACH 5" - $40.00, 8" - $50.00

DUB from S.E. Media -- A UniFLEX BASIC decompiler Re-Create a Source Listing from UniFLEX Compiled basic Programs. Works w/ ALL Versions of 6809 UniFLEX basic.

U - $219.95

LOW COST PROGRAM KITS from Southeast Media The following kits are available for FLEX, SK*DOS on either 5" or 8" Disk.

1. **BASIC TOOL-CHEST $29.95**
   BLISTER.CMD: pretty printer
   LINEXREF.BAS: line cross-referencer
   REMPAC.BAS, SPCPAC.BAS, COMPAC.BAS: remove superfluous code
   STRIP.BAS: superfluous line-numbers stripper
2. **FLEX, SK*DOS UTILITIES KIT $39.99**
   CATS. CMD: alphabetically-sorted directory listing
   CATD.CMD: date-sorted directory listing
   COPYSORT.CMD: file copy, alphabetically
   COPYDATE.CMD: file copy, by date-order
   FILEDATE.CMD: change file creation date
   INFO.CMD (& INFOGMX.CMD): tells disk attributes &contents
   RELINK.CMD (& RELINK82): re-orders fragmented free chain
   RESQ.CMD: undeletes (recovers) a deleted file
   SECTORS.CMD: show sector order in free chain
   XL.CMD: super text lister
3. **ASSEMBLERS/DISASSEMBLERS UTILITIES $39.95**
   LINEFEED.CMD: 'modularise' disassembler output
   MATH.CMD: decimal, hex, binary, octal conversions & tables
   SKIP.CMD: column stripper
4. **WORD - PROCESSOR SUPPORT UTILITIES $49.95**
   FULLSTOP.CMD: checks for capitalization
   BSTYCIT.BAS (.BAC): Stylo to dot-matrix printerr
   NECPRINT.CMD: Stylo to dot-matrix printer filter code
5. **UTILITIES FOR INDEXING $49.95**
   MENU.BAS: selects required program from list below
   INDEX.BAC: word index
   PHRASES.BAC: phrase index
   CONTENT.BAC: table of contents
   INDXSORT.BAC: fast alphabetic sort routine
   FORMATER.BAC: produces a 2-column formatted index
   APPEND.BAC: append any number of files
   CHAR.BIN: line reader

BASIC09 TOOLS consist of 21 subroutines for Basic09.
6 were written in C Language and the remainder in assembly.
All the routines are compiled down to native machine code which makes them fast and compact.

1. CFILL -- fills a string with characters
2. DPEEK -- Double peek
3. DPOKE -- Double poke
4. FPOS -- Current file position
5. FSIZE -- File size
6. FTRIM -- removes leading spaces from a string
7. GETPR -- returns the current process ID
8. GETOPT -- gets 32 byte option section
9. GETUSR -- gets the user ID
10. GTIME -- gets the time
11. INSERT -- insert a string into another
12. LOWER -- converts a string into lowercase
13. READY -- Checks for available input
14. SETPRIOR -- changes a process priority
15. SETUSR -- changes the user ID
16. SETOPT -- set 32 byte option packet
17. STIME -- sets the time
18. SPACE -- adds spaces to a string
19. SWAP -- swaps any two variables
20. SYSCALL -- system call
21. UPPER -- converts a string to uppercase

For OS-9 - $44.95 - Includes Source Code
See Review in January 1987 issue of 68 Micro Journal

Availability Legends
O = OS.9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS.9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

**** Shipping ****
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600

South East Media

Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

SOFTOOLS

The following programs are included in object form for immediate application. PL/9 source code available for customization.

READ-ME Complete instructions for initial set-up and operation. Can even be printed out with the included text processor.
CONFIG one time system configuration.
CHANGE changes words, characters, etc. globally to any text type file.
CLEANTXT converts text files to standard FLEX, SK*DOS files.
COMMON compare two text files and reports differences.
COMPARE another check file that reports mis-matched lines.
CONCAT similar to FLEX, SK*DOS append but can also list files to screen.
DOCUMENT for PL/9 source files. Very useful in examining parameter passing aspects of procedures.
ECHO echos to either screen or file.
FIND an improve find command with "pattern" matching and wildcards. Very useful.
HEX dumps files in both hex and ASCII.
INCLUDE a file copy program that will accept "includes" of other disk files.
KWIC allows rotating each word, on each line to the beginning. Very useful in a sort program, etc.
LISTDIR a directory listing program. Not super, but better than CAT.
MEMSORT a high-speed text file sorter. Up to 10 fields may be sorted. Very fast. Very useful.
MULTICOL width of page, number of columns may be specified. A MUST!
PAGE similar to LIST but allows for a page header, page width and depth. Adjust for CRT screen or printer as set up by CONFIG. A very smart print dr ver. Allows printer control commands.
REMOVE a fast file deleter. Careful, no prompts issued. Zap, and its gone!
SCREEN a screen listing utility. Word wraps text to fit screen. Screen depth may be altered at run time.
SORT a super version of MEMSORT. Ascending/descending order, up to 10 keys, case over-ride, sort on nth word and sort on characters if file is small enough, sorts in RAM. If large file, sort is constrained to size of your largest disk capacity.
TPROC a small but nice text formatter. This is a complete formatter and has funct ons not found in other formatters.
TRANSLIT sorts a file by x keyfields. Checks for duplications. Up to 10 key files may be used.
UNROTATE used with KWIC this program reads an input file and unfolds it a line at a time. If the file has been sorted each word will be presented in sequence.
WC a word count utility. Can count words, characters or lines.

NOTE: this set of utilities consists of 6 5-1/4" disks or 2 8" disks, w/ source (PL9). 3 5-1/4" disks or 1 8" disk w/o source.
Complete set SPECIAL INTRO PRICE:
5-1/4" w/source FLEX - SK*DOS - $129.95
w/o source - $79.95
8" w/source - $79.95 - w/o source $49.95

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants -- TSC Extended BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.
F, S and CCF, U - $25.00, w/ Source - $50.00

SOLVE from S.E. Media - OS-9 Levels I and II only. A Symbolic Object/Logic Verification & Examine debugger. Including inline debugging, disassemble and assemble. SOLVE IS THE MOST COMPLETE DEBUGGER we have seen for the 6809 OS-9 series! SOLVE does it all! With a rich selection of monitor, assembler, disassembler, environmental, execution and other miscellaneous commands, SOLVE is the MOST POWERFUL tool-kit item you can own! Yet, SOLVE is simple to use! With complete documentation, a snap! Everyone who has ordered this package has raved! See review - 68 Micro Journal - December 1985. No 'blind' debugging here, full screen displays, rich and complete in information presented. Since review in 68 Micro Journal, this is our fastest mover!
Levels I & II only - OS-9 $69.95

DISK UTILITIES

OS-9 **VDisk** from S.E. Media -- For Level I only. Use the Extended Memory capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increÔments up to 960K. Some Assembly Required.
Level I OS-9 obj. $79.95; w/ Source $149.95

O-F from S.E. Media -- Written in BASIC09 (with Source), includes: REFORMAT, a BASIC09 Program that reformats a chosen amount of an OS-9 disk to FLEX, SK*DOS Format so it can be used normally by FLEX, SK*DOS; and FLEX, a BASIC09 Program that does the actual read or write function to the special O-F Transfer Disk; user-friendly menu dr ven. Read the FLEX, SK*DOS Directory, Delete FLEX, SK*DOS Files, Copy both directions, etc. FLEX, SK*DOS users use the special disk just like any other FLEX, SK*DOS disk
O - 6809/68000 $79.95

LSORT from S.E. Media - A SORT/MERGE package for OS-9 (Level I & II only). Sorts records with fixed lengths or variable lengths. Allows for either ascending or descending sort. Sorting can be done in either ASCII sequence or alternate collating sequence. Right, left or no justification of data fields available. LSORT includes a full set of comments and errors messages.
OS-9 $85.00

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CC0 = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media

5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

**** Shipping ****
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

***OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.**

Telephone: (615) 842-4600

South East Media

Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

HIER from S.E. Media - *HIER is a modern hierarchal storage system for users under FLEX, SK*DOS.* It answers the needs of those who have hard disk capabilities on their systems, or many files on one disk - a y size. **Using HIER a regular (any) FLEX, SK*DOS disk (8 - 5 - hard disk)** can have sub directories. By this method the problems of assigning unique names to files is less burdensome. Different files with the exact same name may be on the same disk, as long as they are in different directories. For the winchester user this becomes a must. Sub-directories are the modem day solution that all current large systems use. **Each directory looks to FLEX, SK*DOS like a regular file, except they have the extension** '.DIR'. A full set of directory handling programs are included, makingthe operation of HIER simple and straightforward. A special install package is included to install HIER to your particu ar version of FLEX, SK*DOS. Some assembly required. Install indicates each byte or reference change needed. Typica ly - 6 byte changes in source (furnished) and one assembly of HIER is all that is required. No programming required!

*FLEX - SK*DOS $79.95*

COPYMULT from S.E. Media -- Copy LARGE Disks to several smaller disks. FLEX, SK*DOS utilities allow the backup of ANY size disk to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply insening diskettes as requested by COPYMULT. No fooling with directory deletions, etc. COPYMULT.CMD understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes BACKUP.CMD to download any size "random" type file; RESTORE.CMD to restructure copied "random" files for copying, or recopying back to the host system; and FREELINK.CMD as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.

Completely documented Assembly Language Source files included.

*ALL 4 Programs (FLEX, SK*DOS, 8" or 5") $99.50*

COPYCAT from Lucidata -- *Pascal NOT required.* Allows reading TSC Mini-FLEX, SK*DOS, SSB DOS68, and Digital Research CP/M Disks while operating under SK*DOS , FLEX1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in Modular Source Code (Assembly Language) to help solve unusual problems.

F, S and CCF 5" - $50.00   F, S 8" - $65.00

VIRTUAL TERMINAL from S.E. Media - Allows one terminal to do the work of several. The user may start as many as eight task on one terminal, under *VIRTUAL TERMINAL* and switch back and forth between task at will. No need to exit each one; just jump back and forth. Complete with configuration program. The best way to keep up with those background programs.

O & CCO - obj. only - $49.95

FLEX, SK*DOS DISK UTILITIES from Computer Systems Consultants -- Eight (8) different Assembly Language (w/ Source Code) FLEX, SK*DOS Utilities for every FLEX, SK*DOS Users Toolbox: Copy a File with CRC Errors; Test Disk for errors; Compare two Disks; a fast Disk Backup Program; Edit Disk Sectors; Linearize Free-Chain on the Disk; print Disk Identification; and Sort and Replace the Disk Directory (in sorted order). -- PLUS -- Ten XBASIC Programs including: A BASIC Resequencer with EXTRAs over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs Compare, Merge, or Generate Updates between two BASIC Programs, check BASIC Sequence Numbers, compare two unsequenced files, and 5 Programs for establishing a Master Directory of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC Cross-Reference Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in TSC BASIC, XBASIC, and PRECOMPILER BASIC Programs.

ALL Utilities include Source2 (either BASIC or A.L. Source Code).

F, S and CCF - $50.00

BASIC Utilities ONLY for UniFLEX -- $30.00

COMMUNICATIONS

CMODEM Telecommunications Program from Computer Systems Consultants, I c. -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".

*FLEX, SK*DOS, CCF, OS-9, UniFLEX, 68000 & 6809h*

Source $100.00 - without Source $50.00

X-TALK from S.E. Media - X-TALK consists of two disks and a special cable, the hookup enables a 6809 SWTPC computer to dump UniFLEX files directly to the UniFLEX MUSTANG-020. This is the ONLY currently available method to transfer SWTPC 6809 UniFLEX files to a 68000 UniFLEX system. Gimix 6809 users may dump a 6809 UniFLEX file to a 6809 UniFLEX five inch disk and it is readable by the MUSTANG-020. The cable is specially prepared with internal connections to match the non-standard SWTPC SO/9 I/O Db25 connectors. A special SWTPC S+ cable set is also available. Users should specify which SWTPC system he/she wishes to communicate with the MUSTANG-020. The X-TALK software is furnished on two disks. One eight inch disk contains S.E. Media modem program C-MODEM (6809) and the other disk is a MUSTANG-020 five inch disk with C-MODEM (68020). Text and binary files may be directly transferred between the two systems. The C-MODEM programs are unaltered and perform as excellent modem programs also. X-TALK can be purchased with or without the special cables, but this special price is available to registered MUSTANG-020 users only.

X-TALK Complete (cable, 2 disks) $99.95

X-TALK Software (2 disks only) $69.95

X-TALK with CMODEM Source $149.95

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media

5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

** Shipping **
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600

South East Media

Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

XDATA from S.E. Media - A COMMUNICATION Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc.

U - $299.99

EDITORS & WORD PROCESSING

JUST from S.E. Media -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.

• Now supplied as a two disk set:

Disk #1: JUST2.CMD object file,
*JUS12.TXT PL9 source:FLEX, SK*DOS - CC*
Disk #2: JUSTSC object and source in C:
*FLEX, SK*DOS - OS9 - CC*

The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .sp .ce etc.) Great for your older text files. The C source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p ,u ,y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features none of the others have. This is it!

Disk (1) - PL9 FLEX only- F, S & CCF - $49.95
Disk Set (2) - F, S & CCF & OS9 (C version) - $69.95
OS-9 68K000 complete with Source - $79.95

PAT from S.E. Media - A full feature screen oriented TEXT EDITOR with all the best of "PIE™". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished. "C" source available soon. Easily configured to your CRT, with special config section.

*Regular FLEX, SK*DOS $129.50*
• SPECIAL INTRODUCTION OFFER • $79.95
SPECIAL PAT/JUST COMBO (w/source)
*FLEX, SK*DOS $99.95*
OS-9 68K Version $229.00
SPECIAL PAT/JUST COMBO 68K $249.00

Note: *JUST in "C" source available for OS-9*

CEDRIC from S.E. Media - A screen oriented TEXT EDITOR with availability of 'MENU' aid. Macro definitions, configurable 'permanent definable MACROS' - all standard features and the fastest 'global' functions in the west. A simple, automatic terminal config program makes this a real 'no hassel' product. Only 6K in size, leaving the average system over 165 sectors for text buffer - appx. 14,000 plus of free memory! Extra fine for programming as well as text.

*FLEX, SK*DOS $69.95*

BAS-EDIT from S.E. Media - A TSC BASIC or XBASIC screen editor. Appended to BASIC or XBASIC, BAS-EDIT is transparent to normal BASIC/XBASIC operation. Allows editing while in BASIC/XBASIC. Supports the following functions: OVERLAY, INSERT and DUP LINE. Make editing BASIC/XBASIC programs SIMPLE! A GREAT time and effort saver. Programmers love it! NO more retyping entires lines, etc. Complete with over 25 different CRT terminal configuration overlays.

*FLEX, CCF, SK*DOS $39.95*

SCREDITOR III from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

*6800 or 6809 FLEX, SK*DOS or SSB DOS, OS-9 - $175.00*

SPELLB "Computer Dictionary" from S.E. Media -- OVER 150,000 words! *Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX, SK*DOS UCS).* Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of Small Disk Storage systems.

F, S and CCF - $129.95

STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/SK*DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.

NEW PRICES 6809 CCF and CCO - $99.95,
F, S or O - $179.95, U - $299.95

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

•• Shipping ••
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600 **South East Media** Telex: 510600630

OS-9, UniFLEX, FLEX, SK*DOS

STYLO-SPELL from Great Plains Computer Co. – Fast Computer Dictionary. Complements Stylograph.
NEW PRICES 6809 CCF and CCO - $69.95,
F, S or O - $99.95, U - $149.95

STYLO-MERGE from Great P ains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.
NEW PRICES 6809 CCF and CCO - $59.95,
F, S or O - $79.95, U - $129.95

STYLO-PAK --- Graph + Spell + Merge Package Deal!!!
F, S or O - $329.95, U - $549.95
O, 68000 $695.00

MISCELLANEOUS

TABULA RASA SPREADSHEET from Computer Systems Consultants – TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires TSC's Extended BASIC.
F, S and CCF, U - $50.00, w/ Source - $100.00

DYNACALC – Electronic Spread Sheet for the 6809 and 68000.
F, S, OS-9 and SPECIAL CCF - $200.00. U - $395.00
OS-9 68K - $595.00

FULL SCREEN INVENTORY/MRP from Computer Systems Consultants – Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires TSC's Extended BASIC.
F, S and CCF, U - $50.00, w/ Source - $100.00

FULL SCREEN MAILING LIST from Computer Systems Consultants – The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires TSC's Extended BASIC.
F, S and CCF, U - $50.00, w/ Source - $100.00

DIET-TRAC Forecaster from S.E. Media – An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate. Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size. Activity Level and Basal Metabolic Rate for normal individua are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.
F, S - $59.95, U - $89.95

CROSS ASSEMBLERS

TRUE CROSS ASSEMBLERS from Computer Systems Consultants -- Supports 1802/5, Z-80, 6800/1/2/3/8/11/HC11, 6804, 6805/HC05/146805, 6809/00/01, 6502 fami y, 8080/5, 8020/1/2/35/C35/39/40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems. Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text.
*68000 or 6809, FLEX, SK*DOS, CCF, OS-9, UniFLEX*
any object or source each - $50.00
any 3 object or source each - $100.00
Set of ALL object $200.00 - w/source $500.00

XASM Cross Assemblers for FLEX, SK*DOS from S.E. MEDIA -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar TSC Macro Assembler Command Line and Source Code format, Assembler options, etc., in providing code for the target CPU's.
*Complete set, FLEX, SK*DOS only - $150.00*

CRASMB from LLOYD I/O -- Supports Motoro a's, Intel's, Zilog's, and other's CPU syntax for these 8-Bit microprocessors: 6800, 6801, 6303, 6804, 6805, 6809, 6811 (all varieties); 6502, 1802/5, 8048 family, 8051 family, 8080/85, Z8, Z80, and TMS-7000 family. Has MACROS, Local Labels, Label X-REF, Label Lengt to 30 Chars. Object code formats: Motorola S-Records (text), Intel HEX-Records (text), OS9 (binary), and FLEX, SK*DOS (binary). Written in Assembler ... e.g. Very Fast.

CPU TYPE - Price each:

For:	MOTOROLA	INTEL	OTHER	COMPLETE SET
FLEX9	$150	$150	$150	$399
SK*DOS	$150	$150	$150	$399
OS9/6809	$150	$150	$150	$399
OS9/68K	------	------	------	$432

CRASMB 16.32 from LLOYD I/O -- Supports Motorola's 68000, and has same features as the 8 bit version. OS9/68K Object code Format allows this cross assembler to be used in developing your programs for OS9/68K on your OS9/6809 com uter.
*FLEX, SK*DOS, CCF, OS-9/6809 $249.00*

GAMES

RAPIER - 6809 Chess Program from S.E. Media -- Requires FLEX, SK*DOS and Displays on Any Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the strongest CHESS programs running on any microcomputer, *estimated USCF Rating 1600+ (better than most 'club' players at higher levels)*
F, S and CCF - $79.95

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

** Shipping **
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

of the castle in quest of a key, fighting bats, rats, armed guards, and whip-cracking henchmen along the way. If you get the key, you then make your way back out through the three rooms. Door Two leads to the four .rooms of the "Fireball" area. You battle mutants and vultures, jump on floating rocks, raft down an underground river, then finally climb masonry in an attempt to reach the wizard and gain control of the fireballs. Through Door Four's "Shield" area, henchmen throw rocks, and dragons breathe fire. If you survive the four rooms, you acquire the Shield and are zapped back to the Great Hall. Door Three's large doors lead you to the Block Knight area of the castle with the three most challenging rooms. The maze-like rope ladders, floating skeletons, and flying gargoyles are all obstacles which stand in the way of toppling the black knight, who hurls his empty ale tankards at you as you try to pull the ring which will unseat him.

You move your realistic "warrior" through the castle by using the keyboard with your left hand: a = left, d = right, s = down, w = up. Other keys allow the hero to kneel, jump, and pick up objects. With the mouse, you can throw rocks and swing the mace. While this may sound complicated, with a little practice it is easy even for a non-typist to use the keys. However, it is NOT easy to do well immediately. It takes practice to learn to maneuver through the various obstacles. In the heat of "battle" you can miss the "up" key and get killed. (An option screen allows you to change the control keys, but the original configuration seems to work best.) People without patience may give up in frustration before they master the skills to pass through the various rooms.

The biggest surprise with Dark Castle was the RealSound™. Actually, I didn't even know that my Macintosh HAD sound beyond the usual beeps until the lightning crashed in the opening screen and the organ began to play Vincent Price movie music. Each room has various sound effects, including rushing water, bat squeaks, and a hero who grunts as he jumps and says "Yeah!" every time he picks up rocks.

The game has no printed manual, relying instead on on-screen instructions. Watching the disk's three demo rooms helps you figure out the best way to handle those rooms, or for even more help, visit your dealer with a blank diskette and beg for a copy of the full demo disk, which will also give you a great idea of the scope of the entire game.

There are three levels of difficulty, beginner, intermediate, and advanced, insuring that the game can remain interesting even after long hours of play. You score points by knocking out the various inhabitants of the castle, and you get bonus points for passing through rooms quickly. A "Scores of Merit" screen keeps track of the top ten players. And should the action become too intense, the tab key pauses everything until you are ready to continue.

With a real arcade game, it takes rolls of quarters to become proficient, then more rolls to become expert. The advantage of Dark Castle is that you pay your "quarters" up front, then practice at your leisure. And hundreds of hours later, the game will still be a challenge (unlike many other computer games). If you enjoy arcade games, and if you have the patience to learn the game skills, Dark Castle is certainly worth the investment.

By: Gloria Anchor

OARK CASTLE by Silicon Beach Software

Programming: Jonathan Gay

Design and Graphics: Mark Stephen Pierce

TEXT HACKING

By:
Daniel R. Killoran, Ph.D
Director /Software
Cognitive Engine Corp
34 Atlantic Street
Lynn, Mass 01902

Are you a "Text Hacker"? Do you get your thrills from manipulating ASCII files to suit your fancy? More soberly, do you get stuck fixing up everybody else's files that this or that program chokes on? Then you are, willy-nilly, a member of the unrecognized, unorganized, and of course unappreciated community of "Text Hackers." *

What do you need to pursue this odd activity? My list of tools consists of:

a) A good editor. (Or two or three; I use six myself, but that's "gilding the lily.")

b) A text processor. The classic ones are TROFF and NROFF and their children and clones. The former is for proportionally spaced text, the latter for fixed (typewriter) spacing. I use TSC's TEXT, which suffices, but could be improved considerably. Frequently (a) and (b) are combined into something called a "word processor" which is all very well, but limits your flexibility in choosing (a).

c) A macro processor. Macro capabilities are provided by all sorts of different utilities, from editors to compilers, but I prefer stand-alone macro processors like the one my company has just introduced for the 6809 - ML/I.

d) SNOBOL, preferably SNOBOL-4. Alas, this isn't yet available for our Motorola-based machines, although it has been implemented on most mainframes and is even available for the IBM-PC.

Most of you are familiar with editors and text processors, and if my experience is any guide, you defend your favorites with an enthusiasm I fear to challenge, which gives me a good excuse to ramble on about macro processors, promoting ML/I in the process.

What is a macro processor? Essentially it is just a systematic text editor, in which the substitutions are specified by some special character patterns, which are processed in the same way as the text to be edited. This is the principal distinction between macro processors and editors which support macro capabilities - the macro commands are imbedded in the text stream rather than being entered from the console.

A good macro processor, however, is much more powerful than an editor, since it can support much more extended capabilities than are convenient to execute from a keyboard. Generally a macro processor is based on a set of *operation macros* which are invoked by the appearance in the text stream of certain text patterns, called "calls" of the operation macros.

The basic operation is text replacement which is arranged by calling an operation macro to define a *user macro* which will thereafter be recognized in the text stream. In ML/I this is done by inserting a string like the following into the text stream.

```
MCDEF ABC AS DEF;
```

After the macro processor encounters this string it will recognize all subsequent appearances of the characters ABC, set off by spaces or punctuation, as *calls* of the user macro ABC, and this string will be *replaced* in the text stream by the string DEF.

This is about the simplest possible example of ML/I's capabilities, yet it exhibits several features of importance.

Note the phrase "set off by spaces or punctuation." This qualification is made because ML/I does not process the input text character by character but in groups of characters called atoms. An atom is either

a) A punctuation character, including spaces and carriage-returns,

or

b) An alphanumeric string bounded on each side by punctuation characters.

ML/I will not, for instance, replace ABC in the string ABCDEFGHIJKLMNO12345XYZ.

Also note the semicolon at the end of the definition. This bounds, or *delimits* the replacement text, and is consequently called a *delimiter*. Specifically, this is the closing delimiter for the macro MCDEF. The other delimiters in the example are "AS" and, for generality, "MCDEF" itself.

Having defined one or more macros, the next facility that the user needs to know about is the ability to "turn off" macro processing. This is done by the feature known as a *skip* which can be defined in the following manner.

```
MCSKIP MT, < > ;
```

Here we have defined the skip whose name is "<" and which is in effect from the appearance of the "<" until the character ">" is encountered. Thus in

```
ABC XYZ <ZZY ABC MNOPQ ABC> GGG ABC
```

only the first and last ABC would be replaced by DEF, e.g.

```
DEF XYZ ZZY ABC MNOPQ ABC GGG DEF
```

Note that the skip characters have disappeared. This is effected by the first argument of the skip definition, the string "MT" which is delimited by the comma. This argument consists of any or none of the letters M, D, and T. (If none is present, you don't need the comma either.) The letter T means that the enclosed text should be copied to the output stream (otherwise it would be deleted.) The letter D means that the delimiters, i.e. the skip name and its terminal delimiter, should be copied onto the output stream. If the letter D had been used in the above definition, the delimiters "<" and ">" would also have been present in the output. The letter M means that this skip must *match* its terminal delimiter, that is, it is allowed to be nested.

Macros can have *arguments*. The macro ABC is an example of a macro without arguments. In ML/I, arguments must be separated by delimiters. Also a special atom or pattern of atoms must be defined as an *insert name*, which when encountered will cause arguments to be inserted into the output text. For example

```
MCINS & . ;
MCSKIP MT, < >;
MCDEF SWITCH , ! AS<&A2. &A1.>;
```

sets up the insert "&", the skip "<", and the macro

"SWITCH" which has two arguments, delimited by a comma and a semicolon. In the output stream, each occurrence of SWITCH will result in the two delimited pieces of text being switched around, e.g.

```
    QQWE SWITCH ZZY, FFG IJK MNP;
```

becomes

```
    QQWE FFG IJK MNP ZZY XXX
```

Note that the spaces around each argument have been eliminated. This is the result of using the insertion "&AI." which trims off all surrounding spaces. If we wanted to retain the spaces, we would have used "&B1." and "&B2." instead.

But enough of textbook examples. Let's examine a real-life problem I encountered just a few weeks ago. I was obliged to try to get a very snooty mainframe to put out a text file in a form I could use on my 6809 Helix. The mainframe was persuaded to yield ASCII, and condescended to produce variable-length records, but by dint of no persuasion would it put carriage returns at the end of each line. So I got a file of 44K characters with no newlines. The Macintosh will edit such files, but most editors on the 6809 have rather conservative notions of how long a line should be, and 44K is rather beyond the bounds of what they have in mind! However, the mainframe also insisted on producing useless line numbers on each record, and I noticed that they ALL ended with four consecutive zeroes. So I processed the file using ML/I and the following definitions:

```
    MCINS & .;
    MCSKIP MT,< >;
    MCDEF SPACES SPACES NO
    AS<MCGO<>L1<>UNLESS<>MCSUB(&A1.,-3,0)=0000;
    &WD0.&L1.&A1.>;
```

which inserted carriage returns where they belonged.

Admittedly, this packs a lot of ML/I's most powerful capabilities into a lot of space, but let's take it one step at a time:

a) The first two lines you have seen before, so they should be no problem.

b) The macro name is "SPACES" which, when it occurs in a macro definition, is a "code word" for "one or more spaces." Hence, every occurrence of a group of one or more spaces in the input text will be considered a call of the macro. The terminal delimiter is also SPACES. Why doesn't it get treated as a call of the macro we are defining? There are two situations to consider:

1) While the macro is being defined.

2) While the macro is scanning text looking for its delimiter.

In the first case, the macro name is not yet in the "dictionary" that ML/I uses to determine if a macro is being called. In the second case, the call is inhibited by the delimiter "SSAS" which indicates that we have defined a "straight-scan" macro, that is one that does not recognize macro calls while scanning for delimiters.

c) The replacement text consists of all text between the delimiters "<" and ">" but this text is subjected to processing by ML/I when it is inserted into the output stream. Consequently, any macro names present in the replacement text are recognized as calls and are expanded/executed at that time. This is why the replacement text was enclosed in a skip - otherwise the macros in the replacement text would have been expanded while the macro SPACES was being defined.

d) MCGO is an operation macro that instructs ML/I to interrupt its scan of text and resume after the indicated "label", provided that the condition specified in the MCGO is satisfied.

e) MCSUB is an operation macro that returns as its value the substring in its first argument from the position given by its second argument to the position given by its third argument. As if that weren't complicated enough, zero and negative numbers indicate offsets from the right end of the text string. What this macro returns is the last 4 characters of the argument of the SPACES macro.

f) The MCGO condition is satisfied UNLESS the last 4 characters of the SPACES arguments are zeroes.

g) If they are zeroes, the effect of the MCGO is to insert a carriage return.

h) In either case, the expression &L1. is a macro "label" and is not copied into the output text.

i) The next item encountered, "&WD0." is just the macro name itself, but specifically it is "the number of spaces in the atom which resulted in the call of the SPACES macro." What this does is insert into the output text the exact number of spaces that were encountered, so that there will be no space suppression when a carriage return is NOT inserted. It doesn't matter much when one is inserted.

j) The final item in the replacement text is "&A1." which just puts the text between the two groups of spaces into the output stream. We would have used "&WA1." if the characters "&" or "<" had occurred in the input file, but they didn't. The "W", which means "Written", would keep the argument from being expanded when it was inserted.

k) The character ">" bounds the replacement text, and the semicolon finishes up the macro definition.

Let's see, have we overlooked anything? Oh yes:

l) The atom "NO" means that the preceeding delimiter (the terminal delimiter SPACES in this case) is an exclusive delimiter, which means that after the macro is expanded, scanning of the input text resumes AT instead of AFTER that delimiter. Since the terminal delimiter is a macro name, it is then expanded. If we didn't do this, only every other group of spaces would be treated as a macro name, and we would certainly miss many of the desired places to insert a carriage return.

m) The parts of the MCGO, which would ordinarily look like

```
    MCGO  L1  UNLESS  MCSUB(&A1.,-3,0)=0000;
```

are separated by the null expression "<>" because otherwise the blanks would be considered calls of the SPACE macro! Now you see why these delimiters must be defined with the "matching" attribute, since otherwise the first occurrence of ">" would terminate the replacement text!

Dazzling, isn't it? And so far we are just in the "muscle-flexing" stage of using ML/I. Let me hear from you if you would like some more tutorials on the use of this program.

EOF

Editor's Note: If anyone qualifies for the title "Text Hacker" it is certainly I! I spend a good portion of each day re-editing someone elses text. Mostly articles, but sometimes technical manuals for one of the foreign or domestic hardware manufacturers who use us to produce their documentation and manuals. Having used ML/I, I only wish that it had been available years ago! ML/I brings a lot of needed power to those of us still using 6809 systems.

DMW

Logically Speaking

The following is the beginning of a continuing series. Most of you will remember Bob from his series of letters on XBASIC. If you like it or want more, let Bob or us know. We want to give you - *what you want!*

The Mathematical Design of Digital Control Circuits

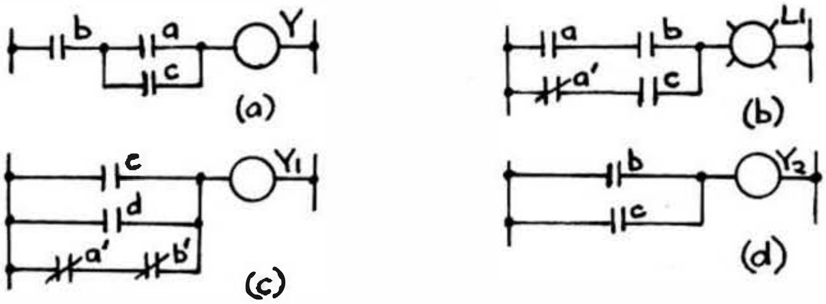
By: R. Jones
Micronics Research Corp.
33383 Lynn Ave., Abbotsford, B.C.
Canada V2S 1E2

Solutions to TEST TWO

```
1.    (a)  Dual = a + bc'          Complement = a' + b'c
      (b)  Dual = ab + c'd'        Complement = a'b' + cd
      (c)  Dual = (a + b')(c' + d) Complement = (a' + b)(c + d')
```

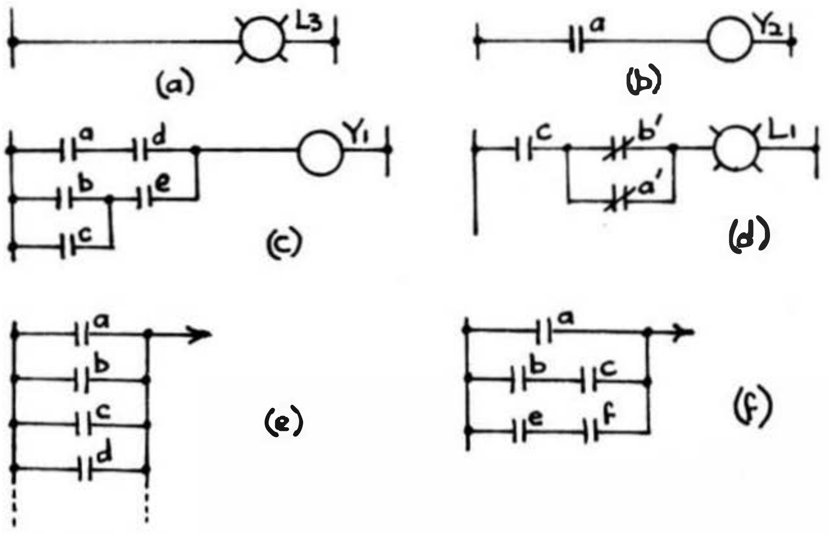
2. Circuit diagrams appear below -

```
(a)  Y1 = ab + bc + abc
        = ab + bc         Rule 5
        = b(a + c)        Rule 7
(b)  L1 = ab + a'c + bcd
        = ab + a'c + (a + a')bcd   Rule 2
        = ab + a'c + abcd + a'bcd  Rule 7
        = ab + a'c                 Rule 5 (twice)
(c)  Y1 = (a + b)(c + d) + a'b'
        = c + d + a'b'             Rules 8 and 6
(d)  Y2 = (a + b)(a' + c) + b + c
        = aa' + ac + a'b + bc + b + c (multiplying out the parens)
        = ac + a'b + bc + b + c    Rule 2
        = b + c                    Rule 5 (3 times)
```



3. To save space, the initial network will not be drawn. This will be left to the student.

```
(a) L3 = x + y + y'  = x + 1  (Rule 2)  = 1  (Rule 3)
(b) Y2 = a + bb'     = a + 0  (Rule 2)  = a  (Rule 4)
(c) Y1 = (a + b + c)(d + e)(b + c + d)(a + e)
       = abc + de + bcd + ae     Form dual
       = a(bc + e) + de + bcd    Rule 7
       = a(bc + e) + d(bc + e)   Rule 7 again
       = (bc + e)(a + d)         and again
       = ad + (b + c)e           Form dual to restore
(d) L1 = a'b'c + ab'c + a'bc
       = b'c(a' + a) + a'bc      Rule 7
       = b'c + a'bc              Rule 2
       = c(b' + a'b)             Rule 7
       = c(b' + a)               Rule 6
(e)      a + a'b + a'b'c + a'b'c'd + .....
       = a + b + b'c + b'c'd + .....  Rule 6
       = a + b + c + c'd + .....      Rule 6
       = a + b + c + d + .....        Rule 6, and so on
(e)      (a + bc + e)(a + bc + f)
       = a + bc + ef             Rule 7
```



Mile 2 - heading for Mile 3

THE BINARY NUMBER SYSTEM

Not again, I can hear you groan! We already know all that stuff! This will be quick and painless, though, unlike our last session, which was pretty tough slogging. But I'm afraid it's necessary to recapitulate our knowledge of binary numbering in order to get the complete picture.

Let's begin by taking a look at the decimal system, in order to understand the relationship between ALL numbering systems in general. We'll concentrate on the simple number 101. What this really signifies is that the rightmost digit specifies the number of "units" in the number, the centre digit the number of "tens", and the leftmost the number of "hundreds". Note that because we are in the DECIMAL system, each digit (as we proceed from right to left) increases by a factor of 10, and that the system also requires 10 symbols in order to function correctly, namely 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Commencing with 0, the set of symbols ends with a digit which is ALWAYS one less than the "base" in which we are working, that is, 10. So, when we read the number 101, we are really adding up "one hundred, no tens, plus 1 unit" to give us our normal "one hundred and one".

Now let's look at binary numbers from the same kind of viewpoint. Here, we are working in base 2, so, commencing with the rightmost digit of any number (or "word" as it's sometimes called), which we read as "units", each digit gets multiplied by the "base" 2 as we proceed leftwards, to give us "twos", "fours", "eights", and so on. Further, because we are working in base 2, we are constrained to TWO symbols only, commencing with 0, and the highest allowable symbol will again be one less than the base, giving us 1. The number 101 in this system is interpreted therefore as "one four, no twos, plus one unit", which is equivalent to 5 in the more familiar decimal system. From all this info, you should readily see that Base 1 would be quite meaningless, as, commencing with the rightmost digit (or units), each digit to the left would increase by a ratio of 1 (thus remaining a "unit" too), and further, the allowable symbols would, commencing at 0, be restricted to a maximum of one less than the base, that is 0 again. So "0" would be all that we'd be able to count.

We are not going to go into binary math here, as it's of no concern in the field of logic, until such time as we need to create a binary-addition or multiplication circuit.

LOOKING AHEAD

In the next section, we're going to learn one of the newer techniques I promised you last time, which is going to make the manipulation of Boolean expressions SO much easier for you, but first let's take a little peek at what the more distant future is going to hold for us. We're going to take our first step away from the direct manipulation of Boolean symbols, and become familiar with translating freely between algebraic terms and their equivalent binary numbers. Then later on in this course, we'll take the ultimate step to the use of decimal numbers, and learn how to design complex control-circuits by "playing around" with these decimal numbers and arranging them in special ways.

We've already tried replacing an uncomplemented symbol with a "1", and a complemented symbol with a "0", when we looked at the Laws of Boolean algebra earlier on, so consider for a moment the following:

In a 2-relay circuit	ab = 11 (binary)	= 3 (decimal)
	a'b = 01	= 1
In a 3-relay circuit	abc = 111	= 7
	ab'c = 101	= 5
In a 4-relay circuit	abcd = 1111	= 15
	abc'd = 1101	= 13

The decimal equivalents are given here only to give you an inkling of how our ideas are going to develop, but for a little while, as I've already said, we'll stick with binary. Just be patient!!

THE KARNAUGH MAP, OR K-MAP

You're now ready to meet a very powerful and versatile ally in the analysis, simplification and synthesis of Digital Control-Circuits. Here he is - the Karnaugh-Map, named after a certain M. Karnaugh. If we are dealing with no more than 5 variables the K-map is practically unbeatable, but beyond this number it becomes increasingly cumbersome to use, as you'll eventually discover for yourselves, but by then we'll have moved on to better things.

I referred to "analysis" and "synthesis" in the preceding paragraph, so let's clarify these words before we go on. Analysis means the study and possible simplification of an already-existing network, whereas "synthesis" means the design and building-up of an original network from a set of specifications.

The basic 4-variable K-map consists of a block of 4 x 4 squares, as shown in Diagram 6, and algebraic terms can be entered on it as shown in each of the maps.

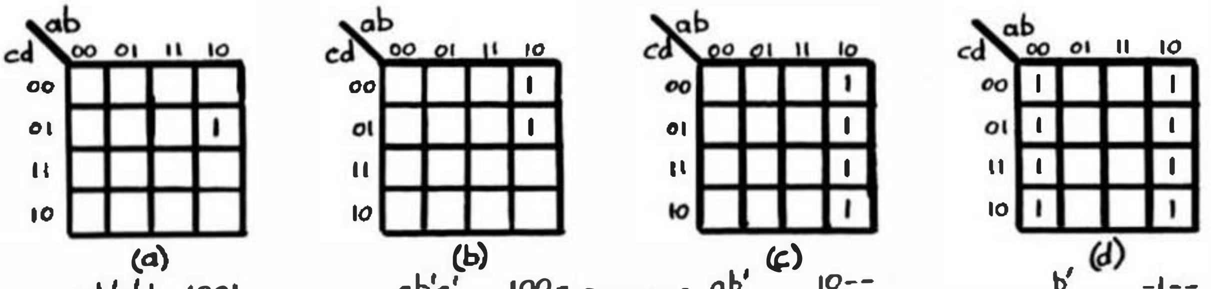


Diagram 6

At the top-left of each square, the letters "ab" and the corresponding column-headings 00, 01, 11, 10 mean that the headings are to interpreted as a'b', a'b, ab and ab'. Similarly with the rows, which are read as c'd', c'd, cd and cd'.

Let's study each map in turn, and establish a system for entering an algebraic term in the correct squares. The first one, ab'c'd is fairly simple - we simply translate it into binary, that is 1001, and enter a "1" where the "ab" column 10 intersects with the "cd" row 01.

Map (b) is a little trickier. Here we must translate ab'c' as 100-, the "-" marking the spot where the literal "d" belongs. Then we enter a 1 in column 10 (the first two binary digits) everywhere that this column intersects with a row where the equivalent "c" is 0, that is, rows 00 and 01. What we've done, in effect, is to enter ab'c' as if it were made up of two separate terms, ab'c'd' and ab'c'd, which, of course, IS so.

$$ab'c'd' + ab'c'd = ab'c'(d' + d) = ab'c' \quad \text{(Rule 2)}$$

Applying these principles still further, in Map (c) we translate ab' as 10--, and simply enter 1 in the 10 column in all rows, as the binary translation tells us to ignore the "cd" row-designations. And finally, in Map (d), we translate b' as -0--, and enter 1 in all columns in which the literal "b" appears as 0 (namely the first and last), and again ignore the "cd" row-designations entirely.

Comparing the four maps, we notice that each time a literal is omitted from a term, the number of 1s entered on the K-map is doubled. Another way of saying this is that each time the number of 1s is doubled (note the relationship of the words "double" and "binary") the corresponding term is reduced by one literal. Note too that the number of 1s is related to the binary system, namely 1, 2, 4, 8 -- there are no terms corresponding to 3, 5, 6 or 7 entries on the map. These observations are VERY important and should never be forgotten. A further point is that even when a block does equate to a binary number it's rectangular in form. You won't find a "1" in one square and a second diagonally situated in a 2-entry block, for instance.

If we look once more at the maps, we see that the only term occupying a single square is ab'c'd, and so it is called ..... ? .... That's correct, it's a "minterm" and now you can see why it's so called. Consider this a moment before reading on. All the clues are contained in the four maps! The answer, of course, is that it is a "minimum term" because it requires the minimum number of 1s to record it on the map. If you try entering a + b + c + d, you'll find that the map will be completely filled up, except for one solitary square somewhere. Hence the name "maxterm", for "maximum term".

TEST THREE

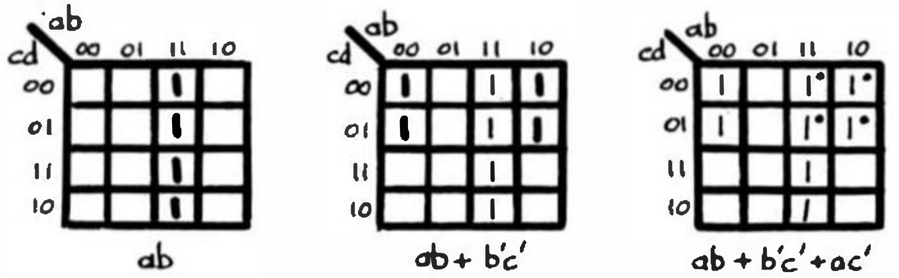
Enter these terms on K-maps. If you brought your pad of squared paper along with you, it shouldn't take long to draw up a set. I've found it a good idea to prepare a whole pageful of these maps and to insert it inside one of those transparent page-protectors which can be picked up almost anywhere. Then you can do your work with a grease-pencil or felt-tip pen, and simply wipe it all off when you're done.

```
(1) a'b'cd'   (2) abc'd'
(3) a'b'c     (4) a'bc'   (5) ab'd    (6) a'cd'    (7) b'cd
(8) ad'       (9) b'c     (10) ab     (11) a'      (12) d'
```

THE K-MAP METHOD OF SIMPLIFYING EXPRESSIONS

From this point, it's but a short step to the simplification of Boolean expressions with the aid of K-maps. Initially, we'll make it a rule to expand our expressions out into sum-of-products form, unless they're already so expressed. Thus (a + b)(c + d) must first be multiplied out into ac + ad + bc + bd.

We'll start off with a very simple example. The problem is to simplify ab + b'c' + ac' on a K-map.



cd\ab	00	01	11	10
00			1	
01			1	
11			1	
10			1	

ab

cd\ab	00	01	11	10
00	1		1	1
01	1		1	1
11			1	
10			1	

ab + b'c'

cd\ab	00	01	11	10
00	1		1•	1•
01	1		1•	1•
11			1	
10			1	

ab + b'c' + ac'

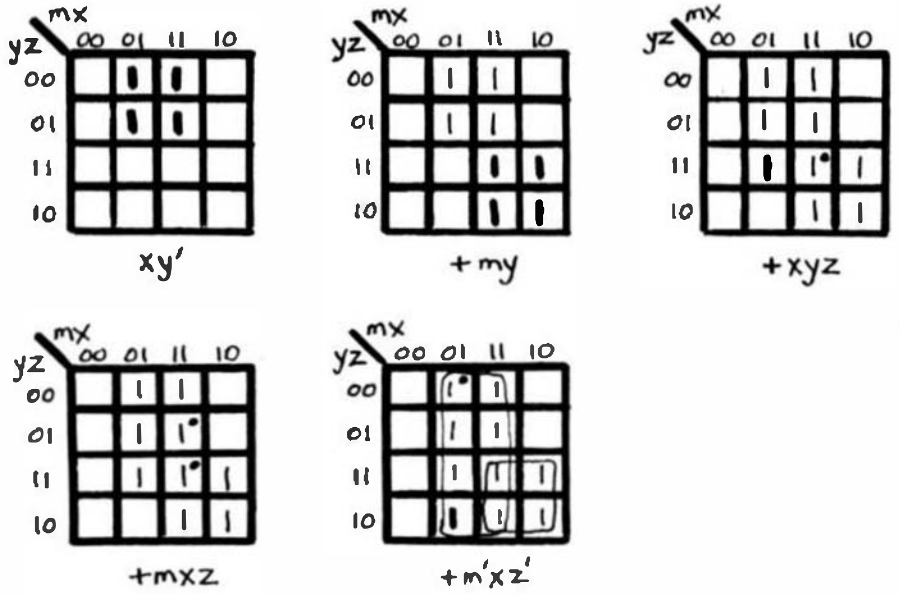
Diagram 7

The successive stages in completing the map are shown above. First, we enter ab, then we add b'c' (note the correspondence between my use of the word "add" and the "+" or OR sign between each term), and finally ac' -- indicated by the squares marked with dots. This shows us right away that ac' is already included in the previous two terms (half of it in each one), so the expression obviously reduces to ab + b'c'. In a case like this, the simplification is immediately clear to us, but this is not always so, and we must learn to "read" a K-map properly before we can say that we're K-map experts. But notice in our example how we were able to do our simplifying without even knowing which Law of Boolean algebra to apply. The K-map did it all for us!!!

Now let's return to our Rules for Club-Membership. Remember how many steps we went through to reduce that one? The rules were at one stage translated into the expression :

xy' + my + xyz + mxz + m'xz'

Each step in the simplification was not too difficult to understand (did I really say that?), but it's still a tough proposition to start from scratch and try to figure which law to apply and when to apply it, bearing in mind that ifyou don't see how to apply a particular law you may not end up with the best reduction! With the K-map, however, it's no problem at all. Simply enter each term on the map, and then read it back out! Here's how :



yz\mx	00	01	11	10
00		1	1	
01		1	1	
11				
10				

xy'

yz\mx	00	01	11	10
00		1	1	
01		1	1	
11			1	1
10			1	1

+ my

yz\mx	00	01	11	10
00		1	1	
01		1	1	
11		1	1•	1
10			1	1

+ xyz

yz\mx	00	01	11	10
00		1	1	
01		1	1•	
11		1	1•	1
10			1	1

+ mxz

yz\mx	00	01	11	10
00		1•	1	
01		1	1	
11		1	1	1
10		1	1	1

+ m'xz'

Diagram 8

The series of 5 maps in Diagram 8 shows how the terms are entered to give us a complete map. First we enter xy', that is -10-, to which is added my, or 1-1-, followed by xyz, or -111, then mxz, or 11-1, and finally m'xz', or 01-0. Notice how some squares are already 1 when we come to make a different entry, but we insert no more than a single "1" in any square. The unoccupied squares are understood to be "0", but we don't bother to fill these in as it makes the map too cluttered and difficult to read.

Knowing that the answer to this problem is x + my, we soon discover that if we enter these two terms on another map we end up with the same map as we have here. Try it as an additional exercise. But if we didn't already know the answer, how do we read this map to give us this m + xy?

READING A K-MAP

Recall for a moment that we found that the larger the block of 1s entered on a K-map, the smaller the term to which it corresponds. So, bearing in mind that one of our chief aims in designing good circuits is to get the smallest possible network to carry out a particular function, we are most definitely interested in the largest possible blocks available on the K-map. Plus, of course, as few blocks as possible to cover all the 1s we have, as long as each block, or "loop" as it's properly called, contains 1, 2, 4, 8, etc ones. It's called a loop because it's customary to draw a loop around each block selected for read-out, as I've done in the final map above.

In reading a map, each "1" MUST be contained in a rectangular loop (no 0s are allowed), but you may include the 1s in as many loops as proves necessary in order to make them as large as possible. In our example I've included two of the 1s, you'll notice, in both loops.

While the method of read-out is quite simple, it still needs a little practice before you become proficient at forming the best combination of loops. Let's try reading the larger loop first. Here's how :

(a) Read the COLUMNS of the loop first, writing down a "-" in each position in which the variable appears in both forms (0 and 1), otherwise write down a 0 or a 1 for the variable.

(b) Apply the same principle to the rows of the map.

Let's elaborate. In the larger loop there are 2 columns involved, namely 01 and 11, and as the first variable appears in both forms, and the second only as "1", we write -1. The loop covers ALL rows, with both "y" and "z" appearing as 0 and 1, so we write down -- for the rows. This gives us the complete term -1--, which translates into "x".

In the snaller loop, only the first "1" is common column-wise, so we write 1- for this. Row-wise, too, it has only the first "1" in common, so again we write 1-, for a combined binary entry of 1-1-, which, of course, translates as "my". The complete read-out is therefore

x + my

and again we have no idea of which Laws got applied, or in which order. Good old K-map did it all for us in one fell swoop!! I know, you're wondering why I didn't tell you all of this earlier, instead of dragging you through the "Laws of Boolean" swamp. Could it be that sometimes the sadist in me gets the better of me? Not really - the swamp happens to be on our route, and without experiemcing the hard parts how can you appreciate to the full the easy parts?

FORMING A GOOD LOOP COVERAGE

Some hints on how to form a good loop coverage of a K-map. ALWAYS start with the smallest loops and form progressively larger ones until every "1" on the map is covered at least once. That is to say, you should first form a loop around all single 1s (one loop apiece, of course) which CANNOT be grouped with any others. That is, there is no other "1" adjacent to it, either horizontally or vertically. Then look for loops which CANNOT be made bigger than two; then four; then eight and so on. Once all the 1s are covered you can stop!!

Loops MUST conform to the binary system (they must contain either 1, 2 4, 8, etc 1s), and also the number of columns or rows in the loop must so conform. No loops spread over 3 columns or 5 rows, for example! Read on a little more before you get too involved with trying your own maps, however.

MAPPING A PROBLEM SOLVES THE PROBLEM

But first make note of the remarkable fact that if you can once convert your initial problem to a form where you can successfully enter it on a map, you have at the same time SOLVED the problem. All you have to do is to read the answer back from the map! And therein, my friend, lies the key to the whole of our circuit-design technique.

MORE ON K-MAPS

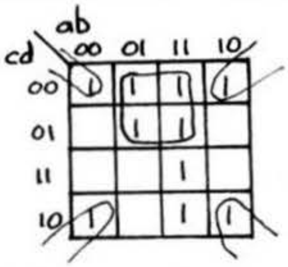


Diagram 9

The K-map in Diagram 9 represents the Boolean expression

ab + bc' + b'd'

Let's go into more detail before we move on to using K-maps to actually design circuits. Maybe you've been wondering why the rows and columns, although labelled with 2-bit binary numbers, are not in numerical order. If they were, they would be in the order 00, 01, 10 and 11. All rather strange, isn't it?

However, if we look a little more closely we notice that commencing on any square on the map, if we move horizontally or vertically by one square, the appropriate column or row designation changes in only ONE of its bits. For example, if we commence in the top-left square, whose coordinates are 0000, and move one square to the right we find ourselves in location 0100. Only the second bit from the left has changed. If we move downwards from this square, only the final bit has changed, to give us 0101, and so on. In other words, *SQUARES WHICH ARE ADJACENT TO ONE ANOTHER CHANGE IN ONLY ONE BIT-POSITION.* It's also true that *SQUARES WHICH DIFFER IN ONLY ONE BINARY BIT-POSITION ARE ADJACENT TO ONE ANOTHER.*

GRAY CODE AND DOUGHNUTS

When binary numbers are arranged in such an order that each successive number changes by one bit only, the ordering is known as a Gray code, and is of PARAMOUNT importance in the designing of RELIABLE digital control-circuits. Even when the sequence overflows from 10 back round to 00, there is still only a 1-bit change! As we're not yet into design, however, let's consider for the moment the consequences of the emphasised statements at the end of the preceding paragraph. One such is that as that the top and bottom rows of our K-map differ in only one bit, they MUST be adjacent. In other words, the K-map is really an opened-out cylinder! But wait!! The left and right-hand columns are only one bit apart too, which leads us to the conclusion that the map really represents the surface of a "toroid" or doughnut.

TRANSMISSION AND HINDERING FUNCTIONS

Now let's come back to Diagram 9, and read it out as follows. The vertical column of four 1s gives us "ab", the circled block of four "bc" and the four corners (which are all adjacent) can be grouped into a block of four to form "b'd'", the complete expression being "ab + bc' + b'd'".

This expression gives us the conditions under which the network as a whole will transmit current to whatever device is connected to it. In words, it says that if Relays A AND B are energised, OR if Relay-B is energised AND Relay-C not, OR if both Relays B AND D are de-energised, the device will be energised. Because it gives the conditions under which the network will transmit current, it is called a *TRANSMISSION FUNCTION.* The blank squares on the map (really 0s) give the conditions under which the circuit will NOT transmit, that is, a'bc + b'd. Do you see how to obtain this? What we did was to group as a loop of 4 the two blanks in the left and right-hand columns, and as a loop of 2 the two blanks in column 01. This is known as a *HINDERING FUNCTION.*

The negation (or complementation, or opposite) of a hindering function is a transmission function. That is (applying it to our current example) :

(a'bc + b'd)' = (a + b' +c')(b + d')

becomes a transmission function, and although this is a completely different network from the original transmission function formed by reading the 1s, the FUNCTION is exactly the same. Draw them both, with a Light as the controlled device, and verify that whenever the relay conditions are such that one of the lights is lit, then so will the other one be also.

A 1s reading always gives us an expression in "sum-of-products" form, while reading the 0s and then negating the expression gives the "product-of-sums" form. The factorised form of an expression, such as the 1s expression above, namely b(a + c') + b'd', is known as a hybrid or "mongrel" form.

VEITCH DIAGRAMS

Prior to M. Karnaugh's K-maps, the columns and rows were arranged in strict binary order, which made it EXTREMELY difficult to figure out which square was adjacent bit-wise to which. Loops also tended to be quite messy as they wiggled and squiggled everywhere to include their 1s, and to avoid those outside the loop. These original maps were called Veitch diagrams (I'm afraid I don't recall the gentleman's initials), and to this day some people STILL use them, erroneously calling them Karnaugh-maps. In their day, though, they represented a step forward from what had gone before.

A FEW WORDS OF COMFORT

Once you've completed TEST FOUR it's fairly safe to say that you're out of the jungle for some time to come, and while I cannot promise that there won't be any more rough patches, I doubt that any will SEEM as tough as these first few miles MUST be to a student having his first encounter with Boolean algebra and K-maps. There ARE more complex ideas ahead of you, but by the time you reach them you'll be a seasoned campaigner and will be well-equipped to deal with them. Nuff said!!

TEST FOUR

Read the following K-maps in both sum-of-products and product-of-sums form.

cd \ ab	00	01	11	10
00		1	1	
01	1			1
11	1			1
10		1	1	

(I)

cd \ ab	00	01	11	10
00	1	1	1	
01	1	1		
11				
10		1	1	

(ii)

cd \ ab	00	01	11	10
00	1	1		
01	1	1	1	1
11	1	1	1	1
10	1	1		

(iii)

cd \ ab	00	01	11	10
00	1			1
01	1	1	1	1
11	1	1	1	1
10	1			1

(iv)

cd \ ab	00	01	11	10
00	1	1	1	
01		1	1	
11	1			1
10	1			1

(v)

cd \ ab	00	01	11	10
00		1	1	1
01			1	
11				
10		1	1	1

(vi)

cd \ ab	00	01	11	10
00	1	1	1	1
01	1			1
11	1			1
10	1	1	1	1

(vii)

cd \ ab	00	01	11	10
00		1		
01	1	1	1	
11	1	1		1
10	1			1

(viil)

As a further test in both completing and reading a K-map, and at the same time simplifying a circuit, you should also try 'TEST TWO again for all examples involving four, or fewer, variables

Next time round, we'll begin designing something or other!! I promise!

..... *End of Mile* 2.

EOF

Pascal

A Tutorial

By: Robert D. Reimiller
Certified Software Corp
616 Camino Caballo
Nipomo, CA 93444
805 929-1359

Last month (actually, August) I mentioned the debugger in passing. This month we will take a closer look at debuggers for high level languages.

The amount of debugging assistance a high-level language package offers varies widely. At one end, there are those packages that offer no support at all (this is typical of many "C" compilers), at the other end there are those that are very good, including being able to manually execute statements any time you want (this is typical of many Basic interpreters).

The debugger I will discuss lies someone in between these two extremes.

First of all, lets see what we need as far as getting the information in to the debugger. The OmegaSoft Pascal compiler can generate three basic output formats for object code :

1) Relocatable object code. Used when we want to link the modules together to get a stand-alone module for use under OS-9 or on a target system.
2) Intermediate code. Used with the debugger, with or without debugging information imbedded into the file.
3) Assembly language. Used for documentation purposes.

Format 2, intermediate code, is a sort of binary assembly language. The format is a series of pascal strings with the following format :

1) Length byte - number of bytes in the record.
2) Opcode - one byte that represents debugger information or an assembly language opcode.
3) Size - one byte representing a data size of byte, word, or long for assembly language opcodes.
4) Postbyte - for assembly language opcodes, indicates the addressing mode, such as register direct, auto-increment, etc.
5) Value - zero or more bytes representing the value that goes with the postbyte, such as a indexed register offset, or external label name.

Fields 4 and 5 can be repeated for those opcodes that require two operands.

In addition to assembly language opcodes, there are special opcodes for debugging information. There are two main types, debug, and line. Line records are very simple, they simply tell the debugger that the code that follows starts at the line number specified in the file. This is used to set breakpoints at the start of pascal statements.

A debug records has a number of options :

1) Start of a procedure. Information included is the procedure's internal name (a unique name for each procedure in the module), parameter size, and procedure name.
2) Start of an external procedure. Same information as 1) above.
3) End of a procedure. Information included is the procedure's internal name, and it's name.
4) Variable record. Information included is :
a) variable name
b) symbol table pointer (in the compiler). This field is used for handling referencing a type that is used as part of another type or variable.
c) block name. This is the internal name of the procedure where the variable/type is defined in (globals have a block of -1).
d) Id group. Can either be a type or a variable.
e) storage group. For instance, register, stack, varib external, record offset.
f) storage address. Depends on the storage group, always 4 bytes long. Can be stack offset, absolute address, an internal name, or register number.
g) Type. Can be the actual type or a value indicating it is a parameter or function return, if so, the actual type will follow.
h) Extra information depending on type :

1) For types that defer to another previously defined type, the symbol table address of that type.
2) For longhex/pointers, a byte that tells whether it is a simple longhex, a pointer to a standard type (such as integer), or a pointer to a non-standard type. If it is a pointer to a non-standard type, then there will be the symbol table address of the non standard type.
3) subranges. Two, byte, word, or longword entries that represent the offset and range of the subrange.

4) Strings. One byte that indicates the maximum length of the string.

5) Sets. One byte that indicates the maximum length of the set (in bytes) and then the entry for the base type of the set (this data starts over with item g) above).

6) Arrays. For each index declaration in the array, the type is defined (starting with item g) above). The last thing is the definition of the base type of the array.

7) Devices. Four byte size of the device followed by the definition for the base type of the device.

8) Records. Four byte size of the record, four byte symbol table address of the last field in the record. For each field in the record a new varib debug record follows, including name fields, record offset, etc.

4) Pointer resolution. Since the base type of a pointer can be defined after the pointer to the object, this record binds the pointer to the type it points to.

5) Module. Tells the debugger the name of this module.

6) Program. Tells the debugger the name of this program.

7) Lex. Updates the debugger as to what lexical level (procedure nesting) is being compiled.

Since this information is imbedded within the code being generated, there is no need to create a separate debug symbol file (which wastes time and storage) and makes correlating between symbols and code relatively easy for the debugger.

The compiler can generate an intermediate file with or without debug information. Any file generated without debug information will run real-time under the debugger, since the same code will be executed as if you went through the link step. Code that is generated with debug information will execute the same code, but in addition, will execute a routine at the start of each pascal statement, and at the start and end of each procedure. The amount of overhead depends on whether there are any breakpoints in place, or if options such as statement trace are enabled.

Certain sections of a program can be selected to have debug information on or off. This is particularly necessary for such things as exception and intercept procedures, which must not be delayed by the debugger for any reason.

The other type of file we want to be able to include when we are debugging is assembly language files. For these we allow loading of relocatable object files.

Now that we have an idea of the files we want, lets look at the commands that have been implemented for handling these files. First of all, the debugger has three different modes. When we enter, we are in the "filer" mode. After all files are loaded, we can move to the "pascal" mode. The third mode, "assembly" mode, can be entered from pascal mode, and then you can return to pascal mode from assembly mode. Once you have left filer mode you cannot return, since some of the symbol table information used in filer mode is re-used once you enter pascal mode.

Filer mode allows the following commands :

1) Load relocatable object files. These can be assembly language modules for support routines for your pascal program, or replacement runtime routines.

2) Load intermediate code. Loads the intermediate code into memory, translating it into 68000 object code for execution. If there is debug information embedded in it, it also builds the symbol table for each module.

3) Link. Resolves inter-module references for both the object code, and symbol table information.

4) Check. Does a cross reference check between symbols in each module. For instance, if you had changed the parameter type of a procedure in one module from value to variable, but forgot to change it in the other modules that call that procedure, this command would warn you about it.

5) Status. Displays amount of table space used and remaining, sizes of each module, and inter-module symbol table references.

6) Default I/O. Lets the program being debugged use the same standard I/O handlers as the debugger is using. If you need separate ports for the debugger and the user program, then these must be loaded as relocatable object files and this command not used.

7) Set exception vector. Allows the user program to handle exceptions, following restrictions that the operating system may impose on the user.

8) Enter pascal mode. Enters pascal debugging mode.

In the pascal debugging mode you are concerned primarily with what happens at the pascal source level. There is a command that will display the current pascal source code where you are stopped at, including an indication of which lines actually generate code so that they can be breakpointed at. Facilities are also included so that you will not stop at the breakpointed line until it has executed a certain number of times. There is also the capability to execute a "macro command" when the breakpoint stops. These "macro commands" are simply a list of commands that you type in before you start the program.

The other main item you want to deal with are pascal variables. The advantage of a high-level debugger like this one is the ability to inspect and change variables in their actual format, be they characters, integers, reals, strings, arrays, or records. This was the reason for including such detailed information about variables in the intermediate code. In addition you also want to be able to index into arrays, or select individual fields within a record. This is especially useful for following a linked list structure, where you might want to look at the 3rd record in a linked list, and a field called count in that record :

```
<P> d base^.link^.link^.count
base^.link^.link^.count = 150
<P>
```

Since source code can change frequently, and none of us really wants to wait for a listing (unless you are looking for an excuse to take a nap), many times we do not get a new listing when changes are made. To make it easier to find what line number we want to breakpoint at, there is a facility that prints the starting line number of a procedure, this combined with the source listing facility, makes locating the desired line fairly easy.

For instance, in the prime number test program, we can find the begin of the main program by typing :

```
<P> n program
prime.9
<P> l prime.9
```

Would display lines before and after line 9 in module prime at the top of the screen, as :

```
 5:  var
 6:    flags, array[0 .. size] of boolean ;
 7:    i, prime, k, count : integer register ;
 8:    iter : integer ;
 9*  begin
10*    writeln ('10 iterations') ;
11*    for iter := 1 to 10 do
12:     begin
13*      count := 0 ;
14*      for i := 0 to size do
15*        flags[i] := true ;
```

In order to start the user's pascal program, we enter the GO command. Before starting the program, the GO command will parse the specified command line (if any) and look for standard I/O redirection modifiers. This allows programs that are written as filters to be properly tested using the same redirection facilities as provided by the shell.

There are two types of proceed commands, proceed and trace statements, and proceed to breakpoint. In the first instance, we can select how many statements to execute until we stop. In the second instance we can proceed until we encounter a previously set breakpoint (up to 16), or proceed to a temporary breakpoint.

Statement trace and procedure trace commands are included. The procedure trace command will list out up to 50 procedure entry's and exit's in the order they occurred. Statement trace will list out up to 50 statement numbers in the order they were executed. Statement trace capture must be previously enabled, this avoids the extra overhead if this feature is not wanted.

Assembly mode can be entered at any time and provides a variety of commands to allow debugging at the assembly language level.

A line by line assembler, disassembler is provided. The starting location can be specified as either the actual code address, or as a statement number in a pascal module. Runtime routine names are expanded into their actual name, rather than just displaying an address, to make correlation between the pascal source and assembly language more apparent. This mode if useful when you want to "patch" the code rather than going back and editing and compiling a module to make a change. It is also useful to see what the actual code generated was, just in case you are unsure of what the pascal statement is really supposed to do.

Assembly mode also provides it's own breakpoints (up to 16), and again, a macro command can be executed when the breakpoint is encountered. Assembly mode also provides the following commands :

1) Block memory move.
2) Calculate value of expression, works as a simple hexadecimal calculator for figuring offsets.
3) Change registers, both data and address.
4) Dump memory in hexadecimal and ASCII.
5) Find one to 16 byte pattern in memory.
6) Initialize an area of memory.
7) Memory modify/set. Allows inspecting/modifying memory as bytes, words, or longwords. In addition allows skipping every other byte for I/O ports. The memory set command is the same except it does not read the memory contents to allow access to write-only I/O ports.
8) Set offset. This value is used when working with code only, such as disassembly and breakpoints. Normally set to the starting address of the module you are working with.
9) Proceed to breakpoint, add temporary breakpoint if wanted.
10) Display registers.

The target debugger (which runs over a serial link between the host and the target system) also allows assembly language tracing, since it does not have to fight the operating system for the trace vector.

These capabilities provide a quite usable debugging capability without requiring any special hardware attachments, which many of us cannot afford anyway. As a large project progresses, only a few modules will probably be changed at any one time, so that all others can be compiled without debugging information, allowing real-time execution of those modules.

In the next chapter we will take a look at a program that does OS-9 system calls.

OmegaSoft is a registered trademark of Certified Software Corporation, OS-9 and OS-9/68000 are trademarks of Microware Systems Corporation.

EOF

Bit-Bucket

By: All of us

"Contribute Nothing - Expect Nothing", DMW '86

NEW FAMILY OF INDUSTRIAL MICROCOMPUTERS PERFORMS IN HARSH ENVIRONMENT.

GESPAC announces its entry in the systems business with the introduction of a new family of versatile microcomputer systems aimed at the industrial marketplace. The GESCOMP is a powerful real-time, multi-tasking microcomputer system in a rugged and compact single height Eurocard package, that allows the user to monitor, test and automate processes in a variety of harsh industrial and military environments.

The GESCOMP system is intended to be used as a process or cell controller in factory automation applications. The GESCOMP is ideal for use as imbedded computers in intelligent machine tools and specialized instrumentations. The GESCOMP systems can also be used as total hardware and software development workstations.

The GESCOMP offers the highest level of processing power for systems of its class and size. At the low end of GESPAC's offering is the GESCOMP 8300-M which uses a 8 MHz 16-bit 68000 microprocessor and comes equipped with two 1 Megabyte 3.5" floppy disk drives and 512 Kilobytes of RAM. The most powerful member of the GESCOMP family is the GESCOMP 8340-P/HF which features a very high performance 32-bit 68020 microprocessor running at 16.7 MHz with a 68881 arithmetic co-processor, 2.5 Megabytes of RAM, 1 Megabyte of 3.5" floppy disk storage and 40 Megabytes of hard disk storage. Several variations of these systems are offered by GESPAC depending on the amount of processing power needed and the required disk/RAM storage capacity. The open G-64 bus architecture of the systems allows memory expansions up to 32 Megabytes.

The GESCOMP system is supported by the OS-9 disk operating system which is well accepted and familiar to the automation industry. OS-9 features an advanced "UNIX-Like" structure and form. A special library allows OS-9 to run programs written for UNIX in C. Unlike UNIX, however, OS-9 features a lean and fast modular structure that can easily be put into ROM for diskless systems. The real-time, multi-tasking nature of OS-9 permits the user to divide his applications into several concurrent tasks, while allowing real-time response to outside events. The modular, multi-user architecture of OS-9 permits the addition of a new user to the system in no time. Depending on the load of work, the GESCOMP system will support up to 16 users on line simultaneously.

The GESCOMP is delivered complete with the disk operating system, a screen editor, macro assembler, symbolic debugger, linker, and a C compiler. Other high level language interpreters or compilers are optionally available for Fortran, Forth, Basic and Pascal.

A key feature of the GESCOMP system is its open G-64 bus architecture which allows a degree of customization which is unsurpassed in the industry. This customization is essential in order to meet the specific requirement of most industrial applications. For instance, a GESCOMP may be configured as a data acquisition system by adding analog I/O modules and additional nonvolatile storage memory. Or, by adding motor controller cards and industrial I/O interfaces, the GESCOMP can be the main controlling unit of a milling machine.

A GESCOMP system typically provides 8 unused slots, each of which can accept any one of the 150 G-64 board level products in GESPAC's catalog. The GESCOMP will also accept G-64 bus board level products from any of the more than 30 independent G-64 bus vendors. The G-64 bus is a second generation microcomputer architecture aimed at midrange industrial applications.

GESCOMP are designed to communicate with the operator with a standard CRT terminal. It is also possible to connect graphics cards into the system to display high resolution pictorial information of 640 by 480 pixels to 1024 by 1024 pixels in 256 colors out of a palette of 262,144. The user can easily install one or more of these graphics controllers into the same backplane to control multiple displays at once.

Another unique feature of the GESCOMP systems is their networking capability. Using GESNET, a proprietary network protocol from GESPAC, up to 50 GESCOMP systems can talk to one another over 3,000 feet of coaxial cable. The Network operates at 800 kilobits per second and achieves very high throughput thanks to its superior CSMA/CA (Collision Sense Multiple Access / Collision Avoidance) arbitration and its Direct Memory Access operation. GESNET's cost per node is a fraction of the typical cost of more publicized networks such as Ethernet or MAP.

A version of the GESNET controller board is to be released by GESPAC for the VME bus. This connection will allow GESCOMP systems to perform as a front end processor with VME based supervisors in a hierarchical, distributed processing architecture. The network is totally integrated into the disk operating system, thus allowing transparent access to all resources. For instance, all graphic screens, disk storage and communications ports are accessible from any processor in the network.

For use in the most severe environments, where mechanical disk drives are not allowed, the GESCOMP can run without a disk. In this mode of operation, the systems can boot from the operating systems and application programs located in EPROM. The GESCOMP is also expandable to use a bubble memory cartridge system for use in these environments. Using GESNET, a disk based GESCOMP located in a clean environment can serve as file server for several ROM based GESCOMP systems on the harsh factory floor.

The GESCOMP system uses a modular architecture with all vital functions readily accessible behind the front panel for easy service and expansion. The system is based on the 100 by 160 millimeter (4" x 6.25") G-64 bus Eurocard and features a rugged DIN 41612 pin-in-socket backplane architecture. The small form factor of the cards and the superior DIN connector, make the GESCOMP particularly resistant to shock, vibration, and corrosion due to airborne contaminates. GESCOMP is packaged in a table enclosure for development or laboratory environment or in a 19" rack for mounting in a NEMA enclosure or directly into the application. Each GESCOMP includes a 200 Watt power supply.

Other versions of the GESCOMP systems, using the 80286 microprocessor and the MS-DOS operating systems are also available from GESPAC.

GESCOMP systems are available now, prices start at $3995 for single quantity orders of the basic system configuration. OEM discounts are available for large quantity orders.

# # #

BILL WEST INCORPORATED
174 ROBERT TREAT DRIVE
MILFORD, CONNECTICUT 06460
203 878-9376

NEWS RELEASE

SYSTEM CONFIGURATION SERVICES

Bill West Incorporated is pleased to announce the availability of system configuration services for users of industrial computers. These services are intended to assist the user in selecting and integrating a system with appropriate interfacing and system software for particular applications. BWI specializes in systems using the VME and STD bus, single board computers, and the OS9 operating system. BWI will develop drivers and software libraries to allow simple control of I/O devices from assembly and high level languages. BWI can supply tested systems configured to customer requirements, with the OS9 operating system installed.

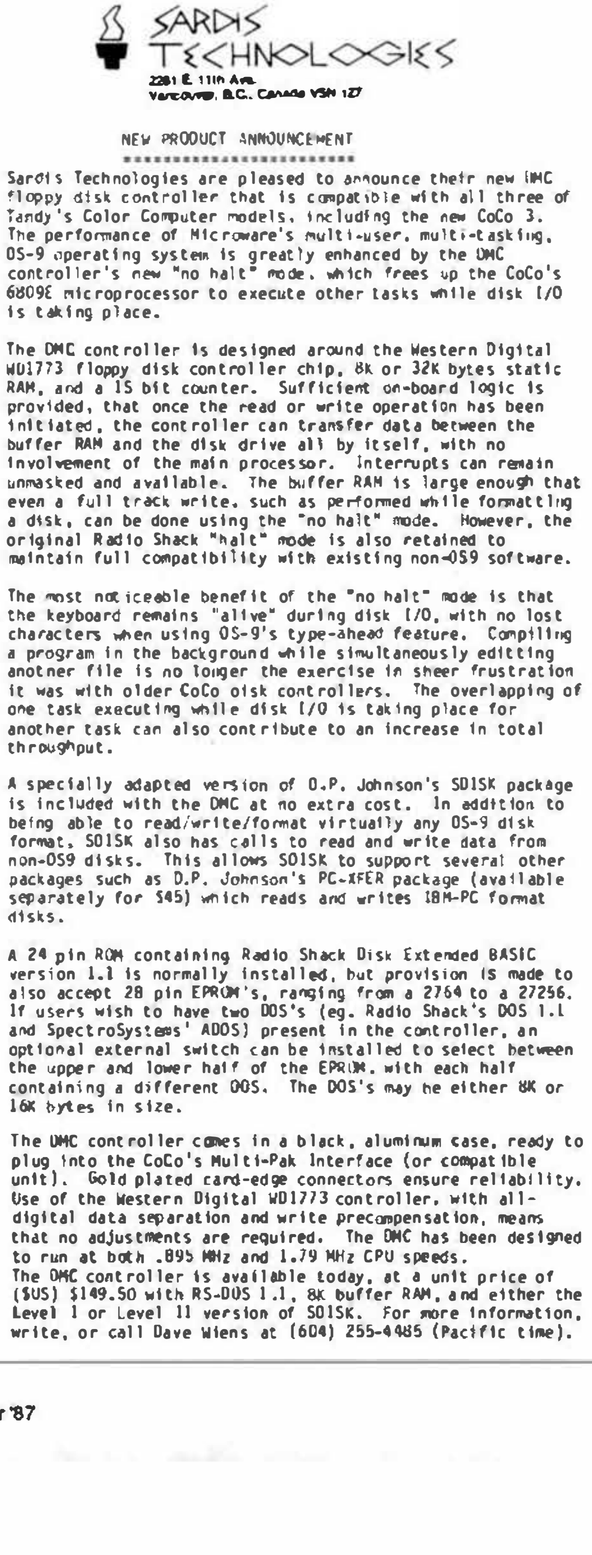
I/O EXPANSION

Bill West Incorporated currently has under development a line of products which allow the use of the STD BUS as an I/O expansion bus for VME, single board, Macintosh, and Atari ST computers. This allows the wide variety of existing STD bus interfaces to be used with these powerful computers. The STD BUS may be configured as a bus extension or as an intelligent subsystem. Communication between the master processor and the STD BUS is via the Motorola I/O channel, SCSI port, or ARCNET interface. Existing STD systems using non-Motorola processors can be incorporated into a larger system using a 68000-type processor as the master processor. These products will be available in the fourth quarter of 1987.

ABOUT BILL WEST INCORPORATED

Bill West Incorporated was founded in 1985 to provide systems and support to users of industrial computers. We have worked with OS9 systems since 1980. In addition to extensive experience in implementing microcomputer systems, BWI personnel have experience in many aspects of system implementation, including analog and digital design, system specification, system documentation, and the integration of minicomputers and microcomputers into large systems.

+++



SARDIS TECHNOLOGIES
2281 E. 11th Ave.
Vancouver, B.C., Canada V5N 1Z7

NEW PRODUCT ANNOUNCEMENT

Sardis Technologies are pleased to announce their new DMC floppy disk controller that is compatible with all three of Tandy's Color Computer models, including the new CoCo 3. The performance of Microware's multi-user, multi-tasking, OS-9 operating system is greatly enhanced by the DMC controller's new "no halt" mode, which frees up the CoCo's 6809E microprocessor to execute other tasks while disk I/O is taking place.

The DMC controller is designed around the Western Digital WD1773 floppy disk controller chip, 8K or 32K bytes static RAM, and a 15 bit counter. Sufficient on-board logic is provided, that once the read or write operation has been initiated, the controller can transfer data between the buffer RAM and the disk drive all by itself, with no involvement of the main processor. Interrupts can remain unmasked and available. The buffer RAM is large enough that even a full track write, such as performed while formatting a disk, can be done using the "no halt" mode. However, the original Radio Shack "halt" mode is also retained to maintain full compatibility with existing non-OS9 software.

The most noticeable benefit of the "no halt" mode is that the keyboard remains "alive" during disk I/O, with no lost characters when using OS-9's type-ahead feature. Compiling a program in the background while simultaneously editting anotner file is no longer the exercise in sheer frustration it was with older CoCo oisk controllers. The overlapping of one task executing while disk I/O is taking place for another task can also contribute to an increase in total throughput.

A specially adapted version of O.P. Johnson's SDISK package is included with the DMC at no extra cost. In addition to being able to read/write/format virtually any OS-9 disk format, SDISK also has calls to read and write data from non-OS9 disks. This allows SDISK to support several other packages such as D.P. Johnson's PC-XFER package (available separately for $45) which reads and writes IBM-PC format disks.

A 24 pin ROM containing Radio Shack Disk Extended BASIC version 1.1 is normally installed, but provision is made to also accept 28 pin EPROM's, ranging from a 2764 to a 27256. If users wish to have two DOS's (eg. Radio Shack's DOS 1.1 and SpectroSystems' ADOS) present in the controller, an optional external switch can be installed to select between the upper and lower half of the EPROM, with each half containing a different DOS. The DOS's may he either 8K or 16K bytes in size.

The DMC controller comes in a black, aluminum case, ready to plug into the CoCo's Multi-Pak Interface (or compatible unit). Gold plated card-edge connectors ensure reliability. Use of the Western Digital WD1773 controller, with all-digital data separation and write precompensation, means that no adjustments are required. The DMC has been designed to run at both .895 MHz and 1.79 MHz CPU speeds.
The DMC controller is available today, at a unit price of ($US) $149.50 with RS-DOS 1.1, 8K buffer RAM, and either the Level 1 or Level 11 version of SDISK. For more information, write, or call Dave Wiens at (604) 255-4485 (Pacific time).

Continued From Last Month

XBASIC Xplained
or
Things you won't find in the documentation

Copyright 1986 by
R. Jones, Micronics Research Corp.,
33383 Lynn Avenue, Abbotsford,
British Columbia, CANADA V2S 1E2
& Computer Publishing, Inc. (CPI) © 1987

MAXIMUM LINE LENGTH

Earlier on I mentioned that the maximum input-line length in XBASIC was 255 characters (not 127 as stated in TSC's article). A 68MJ reader in Australia questions this, as he has tried several configurations of input-lines, and found that (depending on the nature of statements in the line) the maximum acceptable to XBASIC was a variable amount. I think the most he could get was a 149-character line - a long, long way from the supposed 255! Perhaps I should have emphasizes that the *maximum* is 255, which is not to say that you'll *always* get that number per line.

Let me elaborate. XBASIC has two 255-character buffers - one of which (we'll call it 'A') accepts characters as they are entered from the keyboard, and the other (we'll call this one 'B') is used to tokenise your input-line and parse it before storing it into your main program. Now, if you'll cast your mind back to the beginning of this series, you'll recall my example of a few primitive line-entries and their corresponding tokenised lines, and how amazed we were at the disproportionate amount of code that seemed to be generated. Herein lies the key to the '255' problem. If, due to the nature of the line keyed into Buffer A, the tokenising happens to generate less code in Buffer B than there is in Buffer A (and it can happen!) then you can fill up with 255 characters in Buffer A. On the other hand, if Buffer A generates *more* code in Buffer B, then Buffer B determines the limit of 255 at which to cut off tokenising. Hope this makes it all perfectly clear!

RANDOM FILES

A few readers have written me on the subject of Random Files, which seems to indicate that their use gives rise to little difficulties now and then. Perhaps I can best illustrate the most common problem by reproducing the essence of a little program which appeared in the May 1982 issue of 68 Micro Journal. Here's the program :

```
10 PRINT "FILE CREATION PROGRAM"
40 DIM Q$(5)
50 OPEN NEW "CHECKFILE" AS 1
60 FIELD #1, 50 AS Q$(1), 50 AS Q$(2), 50 AS Q$(3),50 AS Q$(4),50 AS Q$(5)
70 FOR I = 1 TO 5
80 Q$(I) = " ..... 50 spaces ..... "
90 NEXT I
100 DF = 200
110 FOR I = 1 TO DF: PUT #1, RECORD I: PRINT' I: NEXT I
150 CLOSE 1: PRINT'
180 PRINT "DONE WITH FILE CREATION"
```

Although this program may *appear* to be quite straightforward, it nevertheless demonstrates an incomplete grasp of random files and how to create them. So, at the risk of repeating stuff which most readers already know, we'll take it from the beginning. But first, try entering and RUNning the program yourself. You should observe the program displaying the opening message, then a 1, followed by a long wait, then a rush of 5 more numbers, another wait, 5 more numbers, and so on till it reaches a count of 200, at which time Line 180's message will be displayed. All well and good, it seems. But, if you have the means to examine the file on disk with DISKEDIT, you'll find 200 records there, filled completely with '00's (NULs) instead of '20's (SPACEs). So what's wrong then???

Let's repeat the above, replacing the 50 SPACEs of Line 80 with a message such as 'Hello World" and pad out to a total of 50 spaces. Again we would find a file of 200 records filled with '00's, even though Line 110's job is to create

200 records, *each of which should consist of 5 sub-records composed of "Hello World"*. Why didn't this happen? OK, random-file tutorial coming up ...

The program is good down to Line 50, where an instruction to open a file named CHECKFILE on Channel 1 is encountered. As a side-note, the file will not actually be opened until the first attempt to GET or PUT a record to it. Line 60 is also OK, where we define each record as being composed of, or FIELDed as, five sub-records of 50 characters each. But, and what a big BUT it is, we then proceed in Lines 70 - 90 to completely cancel our Line.60. You see, you can only put data into, or modify, the various FIELDs by means of the LSET or RSET statements. By simply defining the five 'Q$'s with an ordinary LET (or implied LET), any prior definition using the same variables is effectively wiped out. Line 80 should therefore have read :

```
80  LSET Q$(I) = " ... message ... "
```

without the need of any trailing SPACEs, as the FIELD statement would automatically pad out to the defined length of 50.

Our program is quite workable at this stage, but why does the displayed count go in little spurts? The explanation is that when a random-file is first created, XBASIC allocates only one record to the file. Actually it reserves 3 sectors, but the first two are used for 'housekeeping' purposes, Sector 3 being our actual Record #1. So, when our Line 110 (110 - 140 in the original program) PUTs the first record to the file, all available sectors have been used up, so the FMS section of DOS takes time out to extend the file a little before it can PUT a few more. The original program is, of course, PUTting absolutely nothing into each record - not even a sub-record, due to the cancellation of the FIELD statement - which accounts for the completely NUL file. All that the program is doing is to extend the file a little at a time as the program loops from 1 to DF.

A better approach would have been to extend our file out to 'DF' (that is, 200) records before we begin PUTting anything at all. To extend a random-file, we merely PUT to a non-existent record, and the file will automatically be extended to that limit. Thus :

```
100 DF = 200: PUT #1, RECORD DF
```

It will take a little while to extend to 200 records, but then you should see the numbers 1 to 200 just a-clicking up on the screen with no pauses between.

EPILOGUE

In bringing this discourse to a close, I'd like to raise a subject which should be of interest to a large number of our readers. As most of you know, I've been studying the inner workings of XBASIC for some years now, working out patches for this and that, or modifying it so it can call BEDIT (an XBASIC line-editor) directly from XBASIC itself. In fact, it was my initial patch to allow BEDIT to be called directly from XBASIC with the command 'EDIT' that prompted this series in the first place. All this has really been stop-gap, finger-in-the-dyke patching, so some time ago *I decided to write my own BASIC* - to be called RBASIC ('R' for 'Robert', my first name).

It's reasonably complete now (in fact, it's been undergoing beta-testing for several months), and will almost certainly be so by the time XBASIC XPLANATIONS goes into production. So, though I've obviously had to make it compatible with XBASIC, so you don't have to scrap all your old XBASIC programs, it differs considerably in several major respects. In general it incorporates all those features which for so long I've wished XBASIC had, and at the same time eliminates some minor annoyances, the end result being :

1. It's been written directly in optimised 6809 code, instead of 'warmed-over' 6800 code.

2. It doesn't have any of the known XBASIC 'bugs', such as the erroneous 'Missing Parentheses' or flawed floating-point math.

3. 'I' has been added as a short form of 'INPUT', similar to '?' for 'PRINT'

4. It includes the function 'ARC', so that ARCSIN, ARCCOS and ARCTAN may be implemented, though for compatibility's sake I've had to retain the original ATN function as well.

5. CHAIN allows CHAINing to a file with any extension, though the file to which it CHAINs has to be of the same type as that currently in memory, i.e., either all BAC files, or all BAS files, unless the extension .BAS or .BAC is specifically given, in which case it is possible to switch from one type to another.

6. XOR (Exclusive-OR) logic-operator has been added to allow more sophisticated logic manipulation.

7. APPEND command added to allow another program to be APPENDed to the end of the one currently in memory. This allows library-files of useful .BAS routines to be created and called up as necessary during new program development, for incorporation into the program.

8. LIST has been expanded, not only to allow let's say 'LIST 100-' to LIST from Line 100 through to the end of the program, but it now allows commands such as 'LIST 100,200-300,700 to be implemented. The function of the latter is, I think, self-explanatory.

9. The 'FLEX' command has been replaced by a 'DOS' statement, allowing programs to exit gracefully under program-control. Thus statements of the form '100 IF X > 7 GOTO 300 ELSE DOS' are acceptable to RBASIC.

10. Similarly, the commands 'TRON' and 'TROFF' have been replaced by a single statement 'TRACE'. 'TRACE 1' turns the TRACE-mode ON, 'TRACE 0' turns it OFF, while 'TRACE 125' causes the program to TRACE for 125 lines then turn itself OFF (maximum 254). This makes for a very powerful debugging tool, especially as the TRACEd Line-Numbers are now displayed horizontally, instead of merely one per line. Statements along the lines of the following may thus be inserted into a buggy program for testing :

```
100 IF X > 7 THEN TRACE 200   or   100 IF A$ = B$ THEN TRACE 1
```

11. Most importantly, it includes a built-in line-editor with the following features :

a. Instant recall of an erroneously-entered line which has just been rejected by RBASIC for some syntax error.
b. Editing of any specified program line, ie EDIT 100.
c. Full cursor-control, LEFT, RIGHT, UP and DOWN. Why UP and DOWN, you ask? Just in case you're editing a 255-character line, and wish to do some editing in the middle of the second displayed line.
d. Express RIGHT/LEFT cursor positioning for moving quickly from one end of a line to the other.
e. INSERT, DELETE and OVERLAY, plus SPLIT and MELD. SPLIT will (as its name implies) split a line into two individual lines at the cursor-point, usually at a colon. MELD, on the other hand, will meld, or join together, two successive program lines, with automatic deletion of the second-line's line-number, and its replacement by a colon in the expanded line. This, obviously, is only a brief outline of what can be done with this new Line-Editor.

Plus several other features, such as :

(i) being able to CONTinue after BREAKing on INCH$(0)
(ii) being able to LIST a .BAC file for amusement or informative purposes.
(iii) bringing GOSUB into line with GOTO, in that it doesn't require a mandatory THEN preceding it in IF-THEN clauses. Now the statement 'IF A>B GOSUB 50' is equally acceptable.
(iv) Just loads of other features, including some very powerful debugging aids, which are detailed in RBASIC's accompanying Instruction article.

Recently added is 'PDEL' to allow a command such as PDEL 100-200 to delete the named block of lines (with embedded precautions to protect the User, of course).

The RENUMBER command (now named RENUM) has also been completely re-written, with all known bugs removed. Now it will correctly process lines containing statements such as '100 IF ERL = 75 GOTO 950', and also lines of the form '100 IF X = 7 GOTO 950' where line 950 is non-existent. It will not leave you unaware of the fact that your program will now GOTO a new Line 950 created by the RENUMbering process. Of course, the Precompiler has also had to be re-worked, so it will recognise and process all the new statements of RBASIC. However, this program has not been re-written (only patched) as it appears to have no problems and has remained unchanged for some years, thanks to TSC's policy of producing only quality software.

Keeping in mind that BEDIT, or other similar add-on Line-Editor, eats up lots of valuable memory-space, sometimes as much as between 1500 and 2000 bytes, you may be wondering just how long this new RBASIC is, seeing that it includes the very versatile, and *much more powerful EDIT command*, plus the extras mentioned above, such as ARC, and other features too numerous to detail. Would you believe that in spite of all these additional goodies it's still currently about 1000 bytes shorter than XBASIC? In addition, due to improved algorithms, more efficient coding and so on, it executes a little faster than XBASIC. Time-comparisons indicate a speed-up of approximately 10% to 16%, depending on the function being performed. In response to reader feed-back, we intend to add as many features as possible, before marketing later on this year (1987). So if you've been concerned about some of the limitations of XBASIC, or are considering an 'add-on' line-editor (which only eats up vital memory), you should maybe hold off for a while.

I hope sincerely that you won't take this as a put-down of XBASIC. This has been a wonderful program (one of the best BASICs around, in my opinion), but its days are numbered, especially as TSC has apparently moved on to better things (such as the 68020) and no longer seems to be supporting it. I've given up on just patching it, as the coding was getting a little out of hand! One can only go so far in adding extra floors to a building, then there comes a time when the project has to be scrapped, and a new structuie built from the ground up.

IN CONCLUSION

I'd like to hear from friends old and new if you have any ideas for incorporating into RBASIC, though I'd still like to keep it as compact as I possibly can. My main problem is going to be a shortage of available code-numbers for the extra tokens, so I'll have to give priority to the most generally useful.

So goodbye! I hope you've enjoyed reading this article as much as I've enjoyed putting it all together.

** TSC, where used in this article, refers to Technical Systems Consultants, Inc.

Editor's Note: RBASIC is also being developed for the SK*DOS 68000 series and the MUSTANG-08™ from Data-Comp. We will certainly let you know as soon as it is ready for shipping. Beta testing is in progress now. Also watch for a new C compiler for SK*DOS and the MUSTANG-08. Things are moving!

DMW

EOF

Classifieds As Sunmitted - No Guarantees

AT&T 7300 UNIX PC, UNIX V OS, 1MB Memory, 20 MB Hard Disk, 5" Drive, Internal Modem, Mouse. Best Offer Gets It.

S+ System with Cabinet, 20 Meg Hard Disk & 8" Disk Drive with DMAF3 Controller Board. 1-X12 Terminal $4800.

DAISY WHEEL PRINTERS
Qume Sprint 9 - $900
Qume Sprint 5 - $800.

HARD DISK 10 Megabyte Drive - Seagate Model #412 $275.

3 - Dual 8" drive enclosure with power supply. New in box. $125 each.

5 - Siemens 8" Disk Drive. $100 each.

Tano Outpost II, 56K, 2 5" DSDD Drives, FLEX, MUMPS, $495.

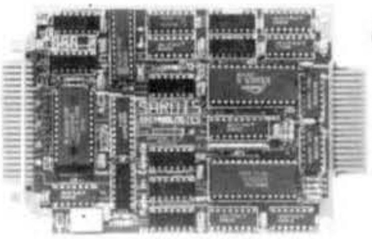
TELETYPE Model 43 PRINTER - with serial (RS232) interface and full ASCII keyboard. $250 ready to run.

SWTPC S/09 with Motorola 128K RAM, 1-MPS2, 1-Paralell Port, MP-09CPU Card - $900 complete.

Tom (615) 842-4600 M-F 9AM to 5PM EST

***

I Want To Buy
#68 DMA FDC. Price and details to
Roger Steedman
RMB 9010
Omeo Hwy Sarsfield
3883 Australia



!!! Subscribe Now !!!
68 MICRO JOURNAL

OK, PLEASE ENTER MY SUBSCRIPTION

Bill My: Mastercard □ VISA □

Card #________________________ Exp. Date_________

For 1 Year ____ 2 Years ____ 3 Years ____

Enclosed: $_____________

Name ____

Street _______________________________________

City ____________________State _____ Zip _________

My Computer Is: _________________

Subscription Rates

U.S.A.: 1 Year $24.50, 2 Years $42.50, 3 Years $64.50

*Foreign Surface: Add $12.00 per Year to USA Price.

*Foreign Airmail: Add $48.00 per Year to USA Price.

*Canada & Mexico: Add $9.50 per Year to USA Price.

*U.S. Currency Cash or Check Drawn on a USA Bank !

68 Micro Journal
5900 Cassandra Smith Rd.
POB 849
Hixson, TN 37343

VISA master card

Telephone 615 842-4600

Telex 510 600-6630

K-BASIC™

The Only 6809 BASIC to Binary Compiler for OS-9
FLEX or SK*DOS
Even runs on the 68XXX SK*DOS Systems*

Hundreds Sold at Suggested Retail:

~~$199.00~~

• 6809 - OS-9™ users can now transfer their FLEX™ Extended BASIC (XBASIC) source files to OS-9, compile with the OS-9 version and run them as any other OS-9 binary "CMD" program. Much faster than BASIC programs.

• 6809 - FLEX users can compiler their BASIC source files to a regular FLEX ".CMD" file. Much faster execution.

• 68XXX - SK*DOS™ users running on 68XXX systems (such as the Mustang-08/A) can continue to execute their 6809 FLEX BASIC and compiled programs while getting things ported over to the 68XXX. SK*DOS allows 6809 programs to run in emulation mode. This is the only system we know of that will run both 6809 & 68XXX binary files.

K-BASIC is a true compiler. Compiling BASIC 6809 programs to binary command type programs. The savings in RAM needed and the increased speed of binary execution makes this a must for the serious user. And the price is now RIGHT!

Don't get caught up in the *"Learn a New Language"* syndrome - Write Your Program in BASIC, Debug It in BASIC and Then Compile It to a .CMD Binary File.

For a LIMITED time save over 65%... This sale will not be repeated after it's over! *

SALE SPECIAL:

$69.95

SPECIAL

Thank-You-Sale

Only From:

CPI

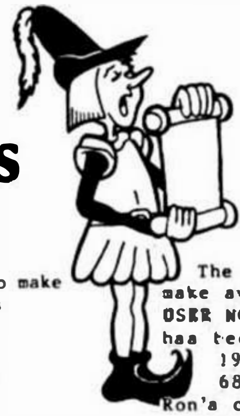
S.E. Media™
5900 Cassandra Smith Rd.
Hixson, Tn 37343
Telephone 615 842-6809
Telex 510 600-6630

A Division of Computer Publishing Inc.
Over 1,200 Titles - 6800-6809-68000

* K-BASIC will run under 68XXX SK*DOS in emulation mode for the 6809.
Price subject to change without notice.

THE 6800-6809 BOOKS

..HEAR YE.....HEAR



OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles
OS9 USER NOTES

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled Operating Programs. The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing your OWN more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

NOTE: .C1,.C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail
or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
• All Currency in U.S. Dollars

Continually Updated In 68 Micro Journal Monthly

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

VISA

(615) 842-4601
Telex 5106006630

*FLEX is a trademark of Technical Systems Consultants
*OS9 is a trademark of Microware and Motorola
*68' Micro Journal is a trademark of Computer Publishing Inc.

OS-9™ SOFTWARE

L1 UTILITY PAK—Contains all programs formerly in Filter kits 1 & 2, and Hacker's kit 1 plus several additional programs. Complete "wild card" file operations, copies, moves, sorts, del, MACGEN shell command language compiler, Disassembler, Disk sector edit utility, new and improved editions, approx. 40 programs, increases your productivity. Most programs applicable for both level I & II 6809 OS-9. **$49.95** ($51.95)

Call or send Self Addressed Stamped Envelope for catalog of software for color ComputerOS-9 and other OS-9 systems.

BOLD prices are CoCo OS-9 format disk, other formats (in parenthesis) specify format. All orders prepaid or COD, VISA and MasterCard accepted. Add $1.50 S&H on prepaid, COD actual charges added.

SS-50C
MEMORY LIQUIDATION SALE!
(While Supply Lasts)

1 MEGABYTE RAM BOARD

Full megabyte of ram with disable options to suit any SS-50 6809 system. High reliability, can replace static ram for fraction of the cost. $399 for 2 Mhz or $439 for 2.25 Mhz board assembled, tested and fully populated.

2 MEGABYTE RAM DISK BOARD

RD2 2 megabytes dedicated ram disk board for SS-50 systems. Four layer circuit board socketed for 2 **Megabytes!** Special sale price of **$399.00** includes only 256k of ram installed (you add the rest), includes OS-9 level I and II drivers for Ram disk, (note: you can reboot your system without losing ram-disk contents). (Add $6 shipping and insurance.)

Please call for answers to your technical questions concerning these products.

D.P. Johnson, 7655 S.W. Cedarcrest St.
Portland, OR 97223, (503) 244-8152
(For best service call between 9-11 am Pacific time.)

OS-9 is a trademark of Microwave and Motorola Inc.
MS-DOS is a trademark of Microsoft Inc.

DATA-COMP
SPECIAL
Heavy Duty Power Supplies

For A limited time our HEAVY DUTY SWITCHING POWER SUPPLY. These are BRAND NEW units. Note that these prices are less than 1/4 the normal price for these high quality units.

5900 Cassandra Smith Rd., Hixson, Tn. 37343
Telephone 615 842-4600 Telex 510 600-6630

Make: Boschert
Size: 10.5 x 5 x 2.5 inches
Including heavy mounting bracket and heatsink.

Rating: in 110/220 volts ac (strap change) Out: 130 watts

Output: +5v - 10 amps
+12v - 4.0 amps
+12v - 2.0 amps
-12v - 0.5 amps

Mating Connector: Terminal strip
Load Reaction: Automatic short circuit recovery
SPECIAL: $59.95 each
2 or more $49.95 each
Add: $7.50 each S/H

Make: Boschert
Size: 10.75 x 6.2 x 2.25 inches

Rating: 110/220 ac (strap change) Out: 81 watts

Outputs: +5v - 8.0 amps
+12v - 2.4 amps
+12v - 2.4 amps
+12v - 2.1 amps
-12v - 0.4 amps

Mating Connectors: Molex
Load Reaction: Automatic short circuit recovery
SPECIAL: $49.95 each
2 or more $39.95 each
Add: $7.50 S/H each

5900 Cassandra Smith Rd., Hixson, Tn. 37343
Telephone 615 842-4600 Telex 510 600-6630

68000 68020 68010
68008 6809 6800

Write or phone for catalog.

AAA Chicago Computer Center

120 Chestnut Lane — Wheeling IL 60090
(312) 459-0450

Technical Consultation available most weekdays from 4 PM to 6 PM CST

Stop!

Get a 25 Mega-Byte Hard Disk practically FREE

only 1¢

*Be Sure to Consider the SPECIAL MUSTANG-08/A 1¢ Sale on outside back cover

*This is exactly one cent more than the price of
the same system - with two floppies - for one cent
more you get one floppy and a 25 MegaByte Hard Disk with the faster
CPU board, additional serial ports and improved clock!
Includes Professional OS-9™ Version 2 and the $500.00 C Compiler!

Remember - When it's over, IT'S OVER!

We don't know how long this very, very low price can be maintained, don't miss it!

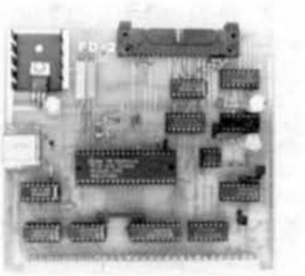
Data-Comp Div. - CPI

IF YOU NEED IT, WE'VE GOT IT!
(OR WE'LL MAKE IT)

Yes, PERIPHERAL TECHNOLOGY still sells the FD-2 for SS-50 Bus Computers. And, if you don't need it, we sell other products from Single Board Computers to Systems which should fit your requirements. Custom Hardware Design is also available. Here's a small sample of what we offer.

FD-2 FLOPPY DISK CONTROLLER

- Controls up to four 5¼" Drives
- Runs in 1 or 2 MHZ Systems
- Can be configured for either 4 or 16 Addresses per I/O Slot (SS30 or SS30C)
- Uses WD2797 Controller Chip (compatiblewith 1771/179X Controller Chip)
- Hardware and Software compatible with SWTPC DC-4 controllers
- 6800/6809 Flex Drivers available
- SK*DOS Operating System or OS9/6809 Driver package

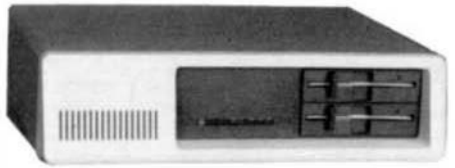


PT69-4 SINGLE BOARD COMPUTER

- 6809E Processor/1 MHZ Clock
- Four RS232 Serial Ports using 6850'S
- Two 8-Bit parallel Ports using 6821 PIA
- Time-Of-Day Clock (MC146818)
- 59K of user RAM
- 2K or 4K of EPROM using 2716 or 2732
- Double Sided/Double Density Floppy Controller
- Can Read/Write Radio Shack OS/9 Diskettes
- Board Size 5.8" X 8.2"

SYSTEMS

- Floppy or Winchester Versions Available
- Systems use any 6809 Single Board Computer or 68008 Board
- Will be configured to meet your Requirements
- OS9 & SK•DOS Operating Systems
- Call or write for system configurations



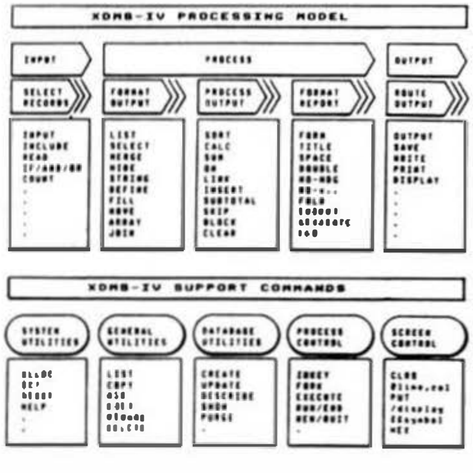
** MONTHLY SPECIAL **
PT68K-1 10 Mhz $350
Regular Price $495

PERIPHERAL TECHNOLOGY
1480 Terrell Mill Road, Suite 870
Marietta, Georgia 30067
(404) 984-0742 Telex # 880584
VISA / MASTERCARD / CHECK / C.O.D.

Catalogues Available Upon Request.

*OS9 Is A Trademark of Microware & Motorola
*SWTPC Is A Trademark of Southwest Technical Systems

XDMS-IV
Data Management System



Save $100.00 - Limited Time
Regular $350.00 ~~- Now Only~~
$249.95

Technical telephone assistance: Tel 914-941-3552 (Evenings)
FLEX™ Tecnical Systems Consultants, SK*DOS™ STAR-KITS Corp.

FOR 6809 FLEX-SK-DOS(5/8")

Up to 32 groups/fields per record! Up to 12 character filed name! Up to 1024 byte records! User defined screen and print control! Process files! Form files! Conditional execution! Process chaining! Upward/Downward file linking! File joining! Random file virtual paging! Built in utilities! Built in text line editor! Fully session oriented! Enhanced forms! Boldface, Double width, Italics and Underline supported! Written in compact structured assembler! Integrated for FAST execution!

XDMS-IV Data Management System

XDMS-IV is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, but also to process entire files producing customized reports, screen displays and file output. Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" on user defined output reports.

POWERFUL COMMANDS!

XDMS-IV combines the functionality of many popular DBMS software systems with a new easy to use command set into a single integrated package. We've included many new features and commands including a set of general file utilities. The processing commands are Input-Process-Output (IPO) oriented which allows almost instant implementation of a process design.

SESSION ORIENTED!

XDMS-IV is session oriented. Enter "XDMS" and you are in instant command of all the features. No more waiting for a command to load in from disk! Many commands are immediate, such as CREATE (file definition), UPDATE (file editor), PURGE and DELETE (utilities). Others are process commands which are used to create a user process which is executed with a RUN command. Either may be entered into a "process" file which is executed by an EXECUTE statement. Processes may execute other processes, or themselves, either conditionally or unconditionally. Menus and screen prompts are easily coded, and entire user applications can be run without ever leaving XDMS-IV!

IT'S EASY TO USE!

XDMS-IV keeps data management simple! Rather than design a complex DBMS which hides the true nature of the data, we kept XDMS-IV file oriented. The user view of data relationships is presented in reports and screen output, while the actual data resides in easy to maintain files. This aspect permits customized presentation and reports without complex redefinition of the database files and structure. XDMS-IV may be used for a wide range of applications from simple record management systems (addresses, inventory ...) to integrated database systems (order entry, accounting...). The possibilities are unlimited.

Visa & Master Card Excepted

Telephone: 615-842-4601 or Telex: 510 600-6630
Or Write: S.E. Media, 5900 Cassandra Smith Rd.,
Hixson, Tenn. 37343

GMX MICRO-20 PRICE LIST

MICRO 20 (12.5 MHz) W/1 SAB	$2565.00
MICRO 20 (16.67 MHz) W/1 SAB	$2895.00
MICRO 20 (20 MHz) W/1 SAB	$3295.00

OPTIONAL PARTS AND ACCESSORIES

68881 12.5MHz Floating Point Coprocessor	$ 195.00
68881 16.67MHz Floating Point Coprocessor	$ 295.00
68881 20MHz Floating Point Coprocessor	$ 495.00
MOTOROLA 68020 USERS MANUAL	$ 18.00
MOTOROLA 68881 USERS MANUAL	$ 18.00

SBC ACCESSORY PACKAGE (M20-AP) $1690.00

The package includes a PC-style cabinet with a custom backpanel, a 25 Megabyte (unformatted) hard disk and controller, a floppy disk drive, a 150 watt power supply, cooling fan, panel mounted reset and abort switches, and all necessary internal cabling. (For use with SAB—9D serial connectors only.)

2nd 5"80 FLOPPY & CABLES FOR M20-AP, ADD	$ 250.00
SECOND 25MB HARD DISK & CABLES, ADD	$ 780.00
TO SUBSTITUTE 50MB HD FOR 25MB HD, ADD	$ 290.00
TO SUBSTITUTE 80MB HD FOR 25MB HD, ADD	$1500.00
TO SUBSTITUTE 155MB FOR 25MB HD, ADD	$2100.00
60MB TEAC STREAMER WITH ONE TAPE	$ 870.00
PKG. OF 5 TEAC TAPES	$ 112.50
CUSTOM BACK PANEL PLATE (BPP-PC)	$ 44.00

I/O EXPANSION BOARDS

16 PORT SERIAL BOARD ONLY (SBC-16S) $ 335.00

The SBC-16S extends the I/O capabilities of the GMX Micro-20 68020 Single-board Computer by adding sixteen asynchronous serial I/O ports. By using two SBC-16S boards, a total of thirty-six serial ports are possible.

RS232 ADAPTER (SAB-25, SAB-9D or SAB-8M) $165.00

The board provides level-shifting between TTL level and standard RS-232 signal levels for up to 4 serial I/O ports.

60 LINE PARALLEL I/O BOARD (SBC-60P) $398.00

The GMX SBC-60P uses three 68230 Parallel Interface/Timers (PI/Ts) to provide up to forty-eight parallel I/O lines. The I/O lines are buffered in six groups of eight lines each, with separate buffer direction control for each group. Buffer direction can be fixed by hardware jumpers, or can be software programmable for bidirectional applications.

PROTOTYPING BOARD (SBC-WW) $75.00

The SBC-WW provides a means of developing and testing custom I/O interface designs for the GMX Micro-20 68020 Single-board Computer. The board provides areas for both DIP (Dual Inline Package) and PGA (Pin Grid Array) devices, and a pre-wired memory area for up to 512K bytes of dynamic RAM.

I/O BUS ADAPTER (SBC-BA) $195.00

The SBC-BA provides an interface between the GMX Micro-20 68020 Single-board Computer and the Motorola Input/Output Channel (I/O bus). With the I/O bus, up to sixteen off-the-shelf or custom peripheral devices (I/O modules) can be connected to the GMX Micro-20.

ARCNET LAN board w/o Software (SBC-AN) $475.00

The SBC-AN provides an interface between the GMX Micro-20 68020 Single-board Computer and the ARCNET modified token-passing Local Area Network (LAN) originally developed by Datapoint Corp. The ARCNET is a baseband network with a data transmission rate of 2.5 Megabits/second. The standard transmission media is a single 93 ohm RG-62/U coaxial cable. Fiber optic versions are available as an option.

OS9 LAN Software Drivers for SBC-AN $120.00

GMX MICRO-20 SOFTWARE

020 BUG UPDATE — PROMS & MANUAL $150.00

THESE 68020 OPERATING SYSTEMS ARE PRICED WHEN PURCHASED WITH THE MICRO-20. PLEASE ADD $150.00 IF PURCHASED LATER FOR THE UPDATED PROMS AND MANUALS. ALL SHIPPED STANDARD ON 5¼" DISKS. 3½" OPTIONAL IF SPECIFIED.

OS9

OS9/68020 PROFESSIONAL PAK $850.00

Includes O.S., "C", uMACS EDITOR, ASSEMBLER, DEBUGGER, development utilities, 68881 support.

OS9/68020 PERSONAL PAK $ 400.00

Personal OS-9 systems require a GMX Micro-20 development system running Professional OS-9/68020 for initial configuration.

Other Software for OS-9/68020

BASIC (included in PERSONAL PAK)	$ 200.00
C COMPILER (included in PROFESSIONAL PAK)	$ 750.00
PASCAL COMPILER	$ 500.00

UNIFLEX

UniFLEX	$ 450.00
UniFLEX WITH REAL-TIME ENHANCEMENTS	$ 800.00

Other Software for UniFLEX

UniFLEX BASIC W/PRECOMPILER	$ 300.00
UniFLEX C COMPILER	$ 350.00
UniFLEX COBOL COMPILER	$ 750.00
UniFLEX SCREEN EDITOR	$ 150.00
UniFLEX TEXT PROCESSOR	$ 200.00
UniFLEX SORT/MERGE PACKAGE	$ 200.00
UniFLEX VSAM MODULE	$ 100.00
UniFLEX UTILITIES PACKAGE I	$ 200.00
UniFLEX PARTIAL SOURCE LICENSE	$1000.00

GMX EXCLUSIVE VERSIONS, CUSTOMIZED FOR THE MICRO-20, OF THE BELOW LANGUAGES AND SOFTWARE ARE ALSO AVAILABLE FROM GMX.

ABSOFT FORTRAN (UniFLEX)	$1500.00
SCULPTOR (specify UniFLEX or OS9)	$ 995.00
FORTH (OS9)	$ 595.00
DYNACALC (specify UniFLEX or OS9)	$ 300.00

GMX DOES NOT GUARANTEE PERFORMANCE OF ANY GMX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

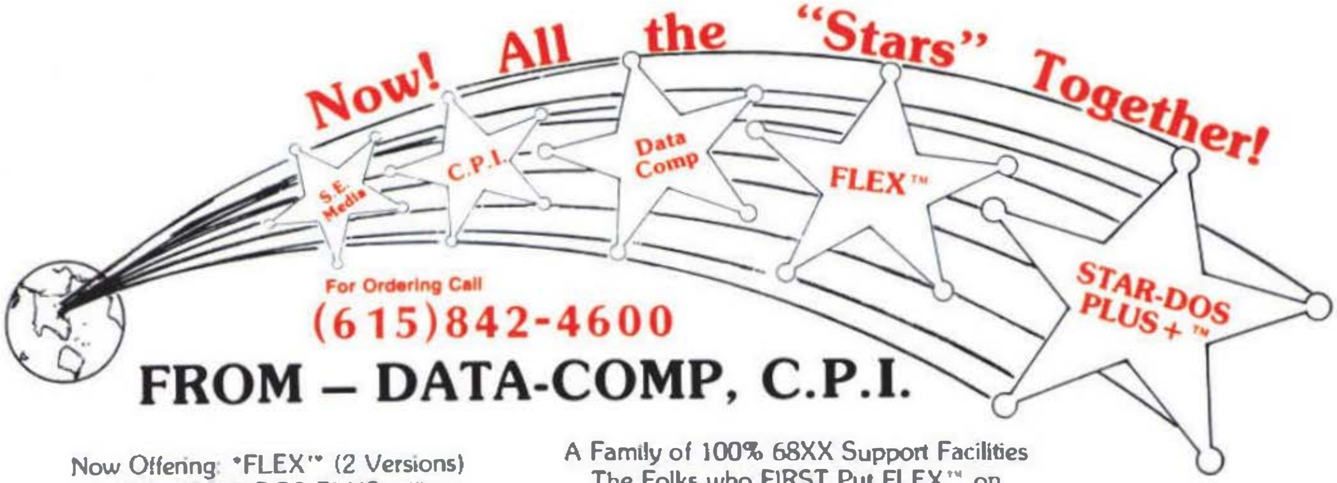
ALL PRICES ARE F.O.B. CHICAGO IN U.S. FUNDS

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

CONTACT GMX FOR MORE INFORMATION ON THE ABOVE PRODUCTS

GMX STILL SELLS GIMIX S50 BUS SYSTEMS, BOARDS & PARTS. CONTACT GMX FOR COMPLETE PRICE LIST.

GMX inc 1337 W. 37th Place, Chicago, IL 60609 (312) 927-5510 — TWX 910-221-4055 — FAX (312) 927-7352



For Ordering Call

(615)842-4600

FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- ● Functions Same as FLEX
- ● Reads - writes FLEX Disks
- ● Run FLEX Programs
- ● Just type: Run "STAR-DOS"
- ● Over 300 utilities & programs to choose from.

$34.95

FLEX-CoCo Jr.
without TSC
Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

- + Read-Write-Dir RS Disk
- + Run RS Basic from Both
- + More Free Utilities
- + External Terminal Program
- + Test Disk Program
- + Disk Examine & Repair Program
- + Memory Examine Program
- + Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY, CABINET, DISK DRIVE CABLE, J&M NEW DISK CONTROLLER JPD-CP WITH J-DOS, RS-DOS OPERATING SYSTEMS.	$469.95

* Specify What CONTROLLER You Went J&M, or RADIO SHACK

THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS	$129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JPD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,F, AND COCO II	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$125.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95

CABLES & OTHER INTERFACES
CALL FOR PRICING

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

☎
(615)842-4600
For Ordering
Telex 5106006630